

AD-A061 158

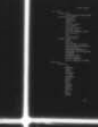
CONSTRUCTION ENGINEERING RESEARCH LAB (ARMY) CHAMPAI--ETC F/G 5/2
SYSTEM DOCUMENTATION FOR COMPUTER-AIDED ENVIRONMENTAL LEGISLATION--ETC(U)
SEP 78 R L WELSH

UNCLASSIFIED

CERL-SR-N-31

NL

1 OF 2
AD A061158



AD A0 611 58

construction
engineering
research
laboratory

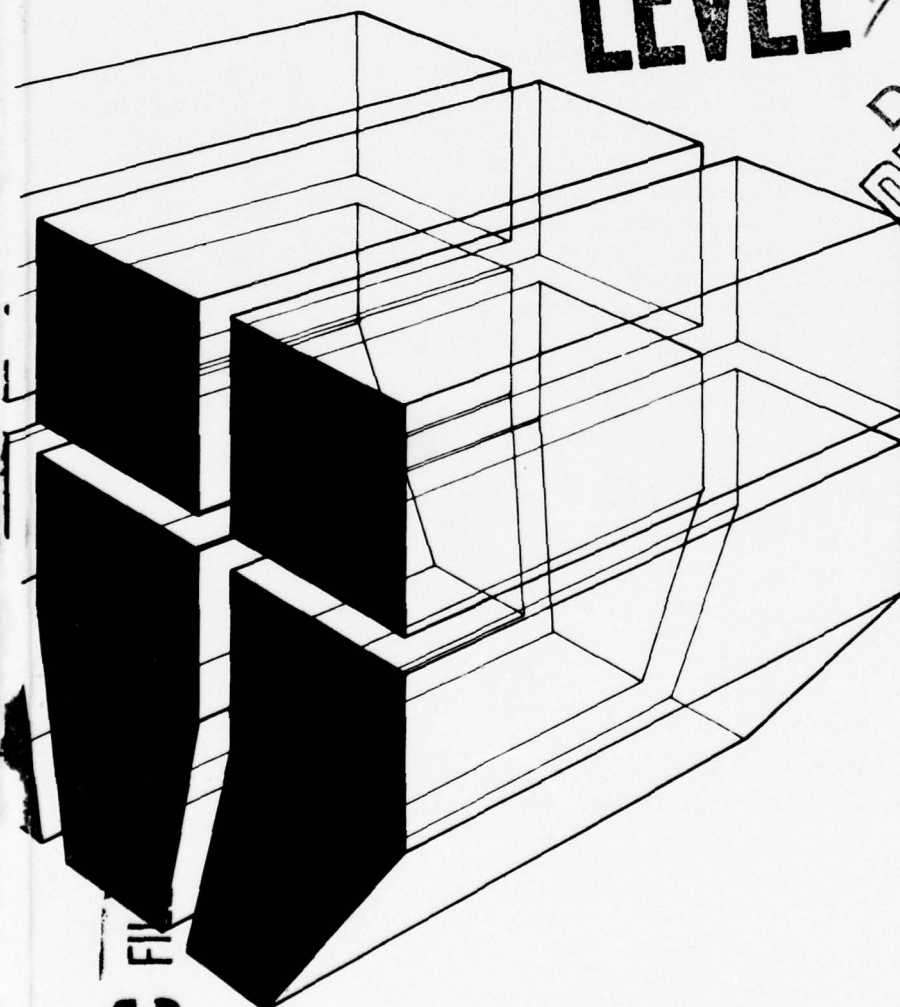
13
P.S.
SPECIAL REPORT N-31
September 1978

SYSTEM DOCUMENTATION FOR COMPUTER-AIDED
ENVIRONMENTAL LEGISLATIVE DATA SYSTEM

LEVEL

DDC
PREPARED
NOV 13 1978
F

by
R. L. Welsh



DDC FILE



78 11 08 011

Approved for public release; distribution unlimited.

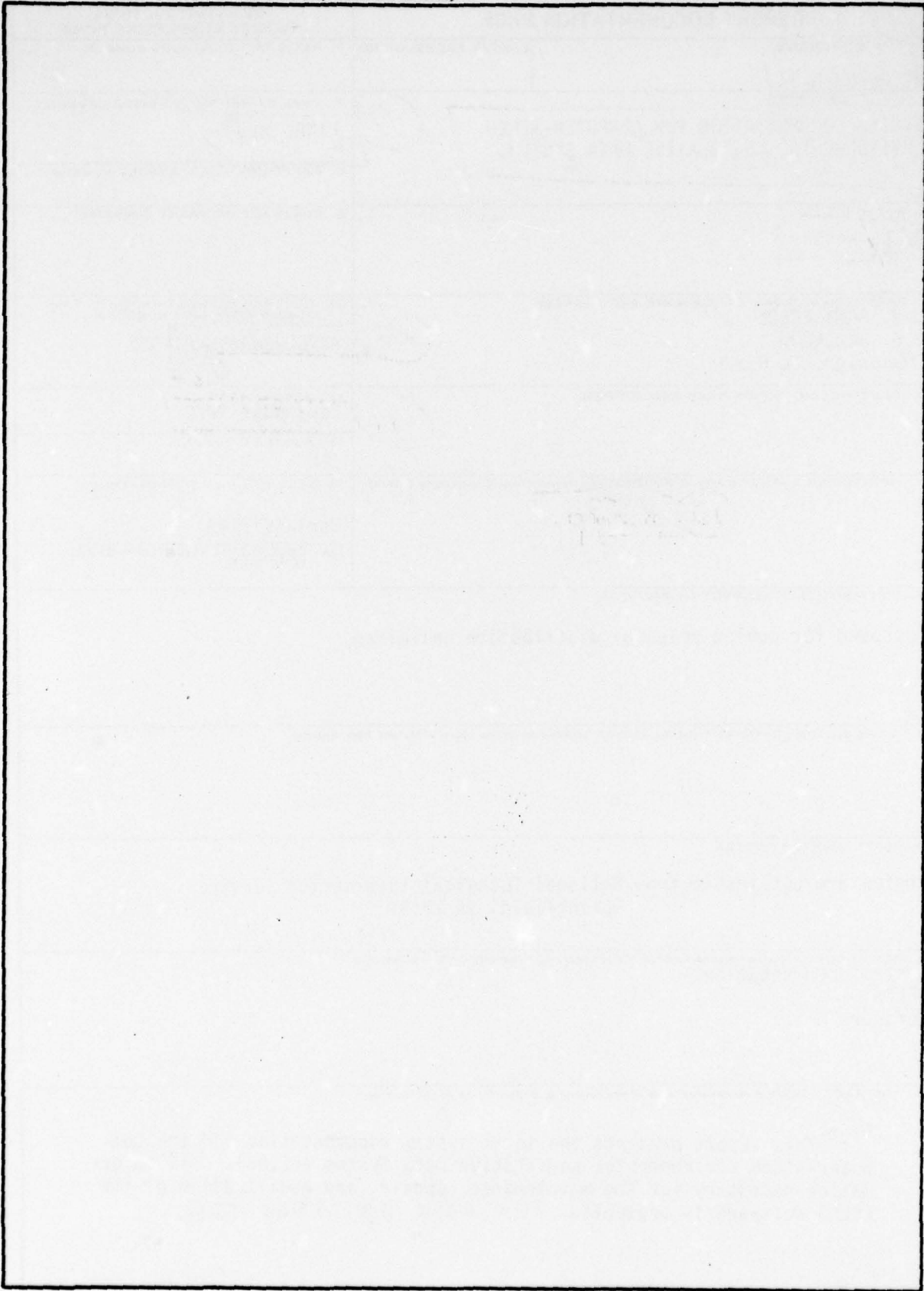
The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official indorsement or approval of the use of such commercial products. The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The program described in this report is furnished by the government and is accepted and used by any recipient with the express understanding that the United States Government makes no warranty, expressed or implied, concerning the accuracy, completeness, reliability, usability, or suitability for any particular purpose of the information and data contained in this program or furnished in connection therewith, and the United States shall be under no liability whatsoever to any person by reason of any use made thereof. This program belongs to the government. Therefore, the recipient further agrees not to assert any proprietary rights therein or to represent this program to anyone as other than a government program.

***DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED
DO NOT RETURN IT TO THE ORIGINATOR***

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CERL-SR-N-31	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SYSTEM DOCUMENTATION FOR COMPUTER-AIDED ENVIRONMENTAL LEGISLATIVE DATA SYSTEM,		5. TYPE OF REPORT & PERIOD COVERED FINAL rept.
7. AUTHOR(s) R. L. Welsh		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. ARMY ECERL P.O. Box 4005 Champaign, IL 61820		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 4A762720A896-01-002
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 159p.		12. REPORT DATE September 1978
		13. NUMBER OF PAGES 159
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Copies are obtainable from National Technical Information Service Springfield, VA 22151		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) system documentation CELDS software		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report presents the total system documentation for the Computer-Aided Environmental Legislative Data System (CELDS). All information necessary for the maintenance, update, and modification of the CELDS software is presented. See also AD-A061 126.		

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

FOREWORD

This project was performed for the Directorate of Military Construction, Office of the Chief of Engineers (OCE), under Project 4A762720A896, "Environmental Quality for Construction and Operation of Military Facilities," Task 01, "Environmental Quality Management for Military Facilities," Work Unit 002, "Development of Environmental Technical Information System." Mr. V. Gottschalk was the OCE Technical Monitor.

This research was made possible through the efforts of Mr. James A. Gast to whom most of the software development is attributed, the Library Research Center of the University of Illinois, and the scientists and engineers of the U.S. Army Construction Engineering Research Laboratory (CERL).

Administrative support and counsel were provided by Dr. R. K. Jain, Chief of the CERL Environmental Division. COL J. E. Hays is Commander and Director of CERL, and Dr. L. R. Shaffer is Technical Director.

ACCESSION for		White Section <input checked="" type="checkbox"/>
		Buff Section <input type="checkbox"/>
NTIS		
DDC		
UNANNOUNCED		
CLASSIFICATION		
BY		
DISTRIBUTION AVAILABILITY CODES		SPECIAL
A		

CONTENTS

DD FORM 1473	1
FOREWORD	3
1 INTRODUCTION.....	5
Background	
Purpose	
Outline of Report	
Description of CELDS Hardware and Software	
Mode of Technology Transfer	
2 DATA COLLECTION PROCEDURES.....	7
Identification	
Collection	
Update	
3 DATA ENTRY PROCEDURES.....	12
Scope of Data Records	
Description of Data	
4 COMPUTER OPERATIONS.....	19
Data Input	
Data Base Creation	
5 DOCUMENTATION OF CELDS PROGRAMS.....	21
CELDS Algorithm	
Detailed File Description	
Documentation of Programs	
6 SUMMARY AND RECOMMENDATION.....	28
APPENDIX A: Attribute Listing	29
APPENDIX B: Keyword Listing	45
APPENDIX C: Source Code and Subroutines	87
APPENDIX D: The Make_Search Subroutine	93
APPENDIX E: The Hier Subroutine	99
APPENDIX F: The Push File	105
APPENDIX G: The Retriever Subroutines	107
APPENDIX H: General Utility Subroutines	147

DISTRIBUTION

SYSTEM DOCUMENTATION FOR COMPUTER-AIDED ENVIRONMENTAL LEGISLATIVE DATA SYSTEM

1 INTRODUCTION

Background

The Computer-Aided Environmental Legislative Data System (CELDS) was developed to respond to the Army's need for rapid, easy access to environmental legislation relevant to a specific project or activity. In 1972, CERL developed a pilot system which contained legislation from six states and the Federal government.¹ Based on results of the pilot, the system's content and operation were modified, and a prototype CELDS containing data for 32 states was implemented in 1975. A user manual² was published in November 1975.

While data were being collected for the remaining 18 states, CERL studied the feasibility of implementing CELDS on a low-cost mini-computer. Results of this study considerably enhanced CELDS, simplified its updating, and significantly reduced its cost.

Purpose

The purpose of this report is to document the current version of CELDS.

Outline of Report

The CELDS documentation will include procedures for selecting and updating legislation to be included in the system (Chapter 2), a description of the data records (Chapter 3), the formatting of legislation into CELDS data records (including abstracting and indexing) (Chapter 4), and complete documentation of the software (Chapter 5).

¹ R. D. Webster, R. L. Welsh, and R. K. Jain, *Development of an Environmental Technical Information System, Interim Report E-52/ADA009668* (Construction Engineering Research Laboratory [CERL], March 1975).

² R. L. Welsh, *User Manual for the Computer-Aided Environmental Legislative Data System, Technical Report E-78/ADA019018* (CERL, November 1975). Superseded by AD-A061 126.

Description of CELDS Hardware and Software

CELDS uses a PDP11/50 minicomputer, stores data on one 88-megabyte disk, and uses the UNIX operating system. All of the CELDS software is written in "C", a high-level language supported by UNIX. The system administrator should have access to and be familiar with the UNIX reference manual³ and the UNIX utilities manual.⁴

Mode of Technology Transfer

The technology transfer will be accomplished in accordance with techniques for computer-assisted systems as defined in appropriate Army regulations.

³ K. Thompson and D. M. Ritchie, *UNIX Programmer's Manual*, 6th ed. (Bell Telephone Laboratories, Inc., May 1975).

⁴ *Documents for Use with the UNIX Time-Sharing System* (Western Electric Company, 1975).

2 DATA COLLECTION PROCEDURES

Identification

Legislation included in CELDS must contain objective criteria pertaining to the environment. The following guidelines should be applied to help identify relevant standards:

1. Laws and regulations containing quantifiable or objective standards should be entered (for example, those which numerically define the maximum permissible amount of a substance which can be released to air or water, or those which name protected species or list specific requirements for the location of a landfill site). "Enabling legislation," which creates or authorizes a specific agency to promulgate and administer regulations, is helpful for understanding the purpose of those regulations, but merits few entries because it does not express concrete, measurable standards. An exception is enabling legislation which also establishes interim standards.

2. Legislation requiring permits or reports for an activity should be entered.

3. Laws and regulations must be originated and administered by a nationwide or statewide agency, i.e., city and county ordinances, are not included.

4. Only *enacted* laws or regulations are included.

5. Laws dealing only with violations of regulations and the penalties for noncompliance are excluded.

Collection

Collection of legislative data is begun by searching administrative codes and/or statutes available in a law library. Administrative codes with complete up-to-date coverage will provide regulations from all agencies and are the preferred source materials for CELDS data records. When a code is available, the index should be searched for regulations pertaining to the following areas:

Air pollution

1. Incineration
2. Industrial plants
3. Refineries
4. Ambient air quality standards
5. Air pollution standards

Drinking water quality standards

Endangered species	<ol style="list-style-type: none">1. Endangered and protected species2. Pests
Erosion	<ol style="list-style-type: none">1. Sedimentation/erosion control
Land Use	<ol style="list-style-type: none">1. Management of coastal wetlands2. Fencing3. Forest cutting4. Dredging5. Landfills
Noise	<ol style="list-style-type: none">1. Motor vehicles2. Aircraft3. Exclude regulations designed to protect employees at work.
Pesticides and economic poisons	<ol style="list-style-type: none">1. Restricted use materials2. Disposal of wastes3. Storage/disposal of containers4. Control of pests
Radiation	<ol style="list-style-type: none">1. Emission limitations2. Exposure standards3. Waste disposal procedures
Solid waste	<ol style="list-style-type: none">1. Disposal of solid and hazardous wastes2. Landfills3. Recycling
Transportation of explosives	<ol style="list-style-type: none">1. Storage and transportation of explosive and hazardous materials
Water pollution	<ol style="list-style-type: none">1. Liquid industrial waste disposal2. Protection of coastal waters3. Oil spill cleanup4. Standards for lakes and streams

Not all states have comprehensive administrative codes, nor do they update them frequently. In these cases, the state statutes are searched in the areas listed above. The statutes occasionally include some laws which meet the CELDS criteria; however, they are primarily useful for providing the "enabling legislation" which gives the names of specific agencies and the activities that each is empowered to regulate. The appropriate agencies should then be contacted for copies of the regulations they administer.

Agency addresses can be obtained from a variety of sources: state blue books, telephone books, and the *Directory of Governmental Agencies Safeguarding Consumer and Environment*.⁵ Agency correspondence is filed, since it often provides corrected or more specific addresses, names of persons to contact, and information about relevant or upcoming regulations. If an agency indicates that certain types of regulations are nonexistent, this fact should be noted and filed.

An alternate source of air quality, water quality, and solid waste disposal agency addresses and regulations is the *Environmental Reporter*.⁶ However, direct correspondence with agencies often produces more current materials and more comprehensive coverage.

Update

Federal

Updating Federal data records that are already in CELDS involves locating changes in, additions to, or deletions from the abstracted legislation. Both substantive and insignificant changes to the scope of CELDS should be considered. Because the effective date on a data record indicates the last noted revision in the legislative contents, this date must be changed to match that of the most recent revision, whether the revision is significant to the data record contents or not. This is done to avoid additional rechecking of a revision that has already been noted. In addition, transfers of administrative agencies, revisions in text, and changes in tables must be noted. Such changes may affect both the bibliographic and legislative reference and the attribute and keyword indexing.

To locate pertinent revisions, additions, or deletions in Federal regulations, the following sources may be used:

1. An up-to-date master list of *Code of Federal Regulations* (CFR) sections included in the Federal data records should be maintained. This list will direct the abstracter to CFR sections that are already abstracted and to the accession number of the data record in which the regulation appears.

2. The *Federal Register* (issued Monday through Friday) contains material affecting existing Federal regulations and also contains newly adopted regulations. Changes in regulations which are already included in the data base may be found by comparing the master list (mentioned

⁵ *Directory of Governmental Agencies Safeguarding Consumer and Environment* (1974).

⁶ *Environment Reporter* (Bureau of National Affairs).

above) with the "Cumulative List of CFR Parts Affected" which appears in each issue. An "affected part" is a section which has been revised, deleted, or added. New regulations, which must also be examined for inclusions in CELDS, may be discovered (1) by checking the cumulative list for section numbers that do not appear on the CELDS master list but are successive to numbers which do appear, and (2) by consulting the contents listing of each issue.

3. Changes cited in the *Federal Register* occasionally refer to a sentence or paragraph in a previously published regulation. The full text of the regulation must then be found to determine what changes have been made. This may involve searching through previous issues of the *Federal Register* or finding the regulation in the *Code of Federal Regulations*.

Since the *Federal Register* is a daily publication, updating CELDS coverage of Federal regulations can be a continuous process, thus insuring optimum currentness. However, updating Federal statutes is a less continuous task because of the nature of the sources used. The primary source is the *United States Code Annotated* (USCA). These volumes are kept current through distribution of cumulative annual pocket parts and monthly pamphlets that contain new laws and judicial constructions. Each volume, pocket part, and pamphlet contains laws, executive orders, proclamations, and an index to the publication's contents. Amendments to statutes already in CELDS can be located in the cited sections of current pocket parts or pamphlets. New laws may be found by using the indexes. The *United States Code* (USC) is the official publication of enacted laws, but is updated less frequently than the USCA and therefore is not useful to this project.

States

Updating of state records follows the general procedures established for Federal regulations. The original sources, such as administrative codes or state statutes, are compared to the existing abstracts for changes in effective date, content, and administrative agency and address.

The individual agencies are requested by letter to provide copies of current regulations and asked to routinely send future changes and additions. Included with the request is a self-addressed prepaid card on which the agency can indicate whether or not regulations are being sent and whether a mailing list is maintained. Agencies which do not have mailing lists are contacted about new or revised regulations at 6-month intervals by postcard. Agencies which do not respond to the initial letter within 3 months are contacted again. As regulations are received from the agencies, the existing abstracts are revised, new laws are abstracted, and all the information is entered into the data base. Legislation pertaining to new CELDS subject areas is found in statutes

and code books; names of possible relevant agencies to be contacted are provided in state blue books or *The National Directory of State Agencies*.⁷

Checking code books and statute supplements against the legislative reference lists reveals any amendments to the legislation. The table of contents of weekly *Environment Reporter* supplements is checked for applicable regulations. Additions, changes, and deletions are checked continuously by examining supplements as they become available, by checking supplements to the *Environment Reporter*, and by contacting state agencies periodically.

⁷ *The National Directory of State Agencies* (Information Resources Press, 1974).

3 DATA ENTRY PROCEDURES

Scope of Data Records

A single agency regulation generally covers many subdivisions of a particular interest area. For example, the Alabama Air Pollution Control Commission's Rules and Regulations booklet contains general sections on provisions for permits, variances, compliance schedules, sampling, records, and reporting. In addition, it contains specific sections on air pollution emergencies, open burning and incineration, and control of various emissions from many emission sources, such as kraft pulp mills, general process industries, nitric acid manufacturing plants, and motor vehicles.

The CELDS abstracter must rewrite this material in concise legal or environmental standards statements for entry into the data base on individual CELDS data records. Each data record should provide information on a specific subdivision of required or prohibited actions, and should be retrievable by a CELDS user.

Description of Data

Each CELDS record consists of 12 data fields:

1. Accession number
2. Title
3. Effective date
4. Legislative reference
5. Major environmental category
6. Geographical/political scope
7. Administrative agency
8. Bibliographic reference
9. Abstract
10. Table of standards
11. Environmental attributes
12. Keywords

Accession Number

An accession number assigned to each data record indicates the order in which it has been collected and entered into the system. The accession number is useful for referring to specific laws in the CELDS retrieval program. If a data record is removed from the system because it has been repealed or amended, its permanent accession number is not reassigned to a new entry, but instead is added to a master list of deleted accession numbers. Accession numbers are entered as digits without any punctuation.

Title

Each data record receives a brief but comprehensive title that indicates the abstract's content. This title helps the user determine the relevance of the entry to his/her specific search. Therefore, it should reflect the scope and emphasis of the abstract, and need not correspond to the heading of the source material. For example, titles for regulations on emission standards from a manufacturing process should include the name of the process and the type of emission, e.g., SULFUR OXIDE EMISSIONS FROM KRAFT PULP MILLS. Each title is followed by a period.

Effective Date

Most laws and regulations are printed with a date or series of dates that indicate when the document or section was enacted, when its contents became law, and when any subsequent revisions or amendments were made. Similarly, the date assigned to a CELDS data record reflects the currentness of the laws from which the abstract is taken, and is generally the most recent date found in the source material. For example, the Water Pollution Control Act of 1972 which was amended December 28, 1973 and January 2, 1974, receives the date 1-2-74. However, there is one exception. When a law which has already been enacted becomes effective significantly later than the enactment date, the enactment date is used in the date field and the effective date is cited in the abstract field. "Significantly later" is more than 6 months. For example, a regulation enacted on November 12, 1974 which will become effective on July 1, 1975 should have the date 11-12-74 assigned to field 3; in the abstract (field 9), the following should be noted: EFFECTIVE 7-1-75, THE FOLLOWING STANDARDS MUST BE MET...).

Accuracy is important when assigning a date to each data record; the CELDS user must know when a regulation or amendment took effect and how current the legislation on a particular subject is. Moreover, this information helps the abstracter keep the data base current, since the legal sources can later be scanned for updating in terms of a predefined cutoff date. When no date appears in the source material, the appropriate government branch should be contacted. Dates are entered numerically, without terminal punctuation, specifically in the order of month-day-year. When a day is not given in the original, the entry is numerical for month-year.

Legislative Reference

The legislative reference is the official source of a law or regulation which tells the user where to locate the full text of an abstract for legal citation purposes. Data record references do not follow strict legal citation format; instead, they use the publication title

followed by a breakdown of as many subdivision numbers and titles as are necessary to enable the user to locate the specific abstracted sections.

Uniformity of citation format is virtually impossible to maintain throughout the records of a given state, or among several states, because different government publishers and agencies tend to develop their own systems of subdivision breakdown. However, for all references to a single publication or to the publications of a single agency, citation format and punctuation should be consistent, conforming to the following general pattern: publication title; chapter number and title; part number and title; subpart number and title; and complete section number (or numbers). Titles are preceded by a colon and followed by a semicolon; the final section number is preceded by a comma and followed by a period. A typical example is:

California Administrative Code; Title 17: Public
Health; Part III: Air Resources; Subchapter I:
Air Basins and Air Quality Standards, Section 70101.

It is often necessary to scan the text of a regulation to determine how it is set up and what terms it uses to refer to various subdivisions. Use of the regulation's own terminology in the legislative reference will prevent confusion to the user who consults the original text.

Major Environmental Category

Assigning major environmental categories is the first step in indexing a data record. There are ten major environmental categories in the CELDS system:

Air Quality	Noise
Earth Science	Sociology
Ecology	Solid Waste
Health Science	Transportation
Land Use	Water Quality

Most data records are assigned to the one category that represents the aspect of the environment most directly affected by the law or regulation; however, regulations may be assigned to as many environmental categories as are applicable. For example, regulations on the use of pesticides and radioactive materials often get assigned to HEALTH SCIENCE, AIR QUALITY, and WATER QUALITY.

Geographical/Political Scope

This field indicates a regulation's political origin and is always a state name, the Federal government, or the District of Columbia. The states are entered by their two-character postal abbreviations. The Federal government is "US," the District of Columbia is "DC," and Puerto Rico is "PR."

Administrative Agency

This field contains the official name and address of the agency responsible for administering a specific law or regulation. If the source of the data record is U.S. or state legislation rather than an agency regulation, the information recorded in this field should be the name and address of the department or agency designated in that particular law to oversee enforcement of its provisions.

Bibliographic Reference

The bibliographic reference indicates the printed source in which a law or regulation was located by the abstractor, and from which photocopies were taken for the CELDS manual files. It is preferable to take data from the original legislative source, because it is more reliable than unofficial reprints. In these cases, field 8 should read "SAME AS LEGISLATIVE REFERENCE." If an agency has sent regulations that are not available in statutes or codes, the bibliographic reference should read: "AGENCY (or DEPARTMENT) PUBLIC INFORMATION PAMPHLET."

For the areas of air and water quality, solid waste, land use, and noise, regulations received from an agency should be checked for accuracy and currentness against the contents of the *Environment Reporter*. If rules and regulations in the *Environment Reporter* are identical and up to date, they should be cited as the bibliographic reference instead of the agency copies.

Use of the *Environment Reporter* is an exception to the rule stated in the previous paragraph, because it is available commercially and through libraries; it is therefore a more convenient source for checking a text than agency reprints which are not readily available. Where the *Environment Reporter* is cited, use the title, volume name, and section number; the parts of the reference should be separated by commas; for example, ENVIRONMENT REPORTER, STATE AIR LAWS, 361. Page numbers should not be included because service is continuously updated and the pagination is therefore temporary. All bibliographic references end with a period.

Abstract

The abstract is a concise, informative presentation of pertinent details in a law or regulation. Its opening sentence should repeat or rephrase the title (field 2). Abstracts must be written in a straightforward narrative style, eliminating verbiage and legal jargon; however, coverage of technical specifications should be thorough and precise. Most source documents include a section of terminology definitions. Since it is assumed that CELDS users are familiar with standard scientific terms and technical terms, these are not generally included in the data base. If a regulation uses a term in an uncommon or specialized

context and a definition is required for clarification, it should be incorporated into the abstract text. When a chart or table is used to present data, the abstract should describe its contents briefly (subject and scope) without detailing the specifics; these will appear in tabular format in field 10.

Individual states regulate different areas of the environment in varying depths and organize their coverage in different ways. The abstracter must determine which sections of a document are relevant to CELDS, and how to present those sections in an organized, concise, and retrievable form. Each data record should be a self-contained unit concerning one or more related aspects of a subject. If the source document is well organized, CELDS coverage may simply follow the subdivision levels in the regulation, with one data record for each division or group of subdivisions. For a more complex or poorly organized document, however, it may be necessary to reorganize the grouping of sections for entry into data records. An air quality regulation may present rules 1a-e for air contaminant emissions from *existing* sources of type A to E, followed by rules 2a-e for emissions from *new* sources of type A to E. If the limits specified for new and existing sources of type A are identical or vary in only some specifics, rules 1a and 2a should be combined in a single data record. Similarly, radiation standards often list exposure limits for "individuals" in one section and for "minors" several sections later. It is not only logical to abstract these rules together in a single data record, but also potentially misleading not to. A CELDS user searching index terms for radiation regulations will find terms differentiating between maximum permissible dose and concentration levels, radioactive wastes, and radiation hazards, but no terms relating to age limits. A user who retrieves a data record on exposure limits for "individuals" may not be aware that he should search further for similar rules on "minors"; therefore, the two rules should be in the same data record. This is also true for regulations concerning "old" and "new" sources of air pollution and for many other sub-topics.

A section may sometimes need to be repeated in more than one data record (for example, a statement of applicability or a definition of exceptions which applies to several data records). In all cases, combinations and repetitions of sections will be reflected in the legislative reference (field 4). When reviewing a source document to determine the breakdown for data records, the abstracter should remember that sections may be combined only if the resulting data records can be indexed distinctively for retrieval and will not require the user to read through lengths of material to find a specific section of interest.

Basic requirements for permits, reports, and tests should be included in the abstract, but without administrative details or test procedures. Abstracts may also include parenthetical or explanatory notes by the abstracter when a source document contains an obvious error or is confusing. References to other documents contained in the CELDS data

base should be avoided; however, when such references are necessary, a citation is given to the appropriate CELDS accession number.

Table of Standards

This field is used when the best way to present the content of a regulation or portion of a regulation is in tabular format (for example, rules on maximum permissible levels of chemical substances in the air or water). A table must use no more than 60 spaces across the page, and must have a table number, descriptive title, and column headings which include the relevant units of measure. Tables are numbered consecutively with each data record. If no table is necessary for a particular accession number, the word "NONE" is entered in this field.

Environmental Attributes

Environmental attributes (EA's) are index terms developed by CERL which are arranged by areas under corresponding major environmental categories. (Appendix A provides the list of acceptable attributes.) The list is arranged hierarchically in three levels: (1) parametric terms form the broadest level; (2) subparametric names list subdivisions of parametric terms; and (3) at the most specific level, detailed attributes list individual chemical compounds, types of plants and animals, and other environmental aspects. Terms from any level of the hierarchy may be assigned to a data record, depending on the level of specificity of the document being indexed. As many terms as necessary should be used to describe the content of the data record adequately; however, no EA term may be assigned unless the corresponding Major Environmental Category (MEC) was entered in field 5. It should be noted that an attribute may be expressed by several terms within one MEC. For example, RADIOACTIVE and RADIOACTIVE EMISSIONS both appear in the EA list under the MEC AIR QUALITY. Similarly, different MEC's may use different terms for related subjects. The EA's RADIATIONS and OTHER RADIATIONS appear under the MEC HEALTH SCIENCE; the EA's RADIOACTIVE and RADIOACTIVITY appear under MEC WATER QUALITY. For a data record on emissions of radioactive wastes to air and water, all of these terms should be listed in field 11. Particular attention should be paid to attributes listed beneath the parametric name CONTROVERSIAL ATTRIBUTES, which appears at the end of each major environmental category section of the EA list. These are frequently variations of general terms and are of particular interest to users.

When EA's are assigned, regulations that are similar from state to state should be indexed consistently so that users searching the system can easily make a transition between states or from the Federal government to a state. EA's should be entered one to a line, without punctuation, and should be uniform. If no EA pertains to a data record, the word NONE is entered in this field.

Keywords

Because an appropriate attribute does not always exist for a particular data record, a thesaurus of keywords has been developed as a supplementary index (Appendix B provides a listing of acceptable keywords). It differs significantly from the list of environmental attributes because of the inclusion of process names (e.g., MANUFACTURING) and procedural terms (e.g., PERMITS) in addition to names of the chemical compounds and environmental variables affected by the processes; this allows an added degree of document separation and content identification. This list is arranged hierarchically by major environmental category.

To assign keywords to a CELDS document, the major environmental categories under which the document belongs must be determined and the keyword list consulted for these specific categories. It should be noted that the keyword thesaurus is dynamic, i.e., it may grow or be revised. It is conceivable that new legislation regulating sectors of the environment that were not considered previously may not fit into the existing structure of keywords (or major environmental categories); thus, they cannot be accurately indexed with the existing terms, and creation of new keywords may be necessary. When this happens, it is also necessary to determine whether any of the past laws already in the data base should have this keyword added to their indexing terms.

4 COMPUTER OPERATIONS

Data Input

When legislation has been collected and the data records established according to the outlined procedures, the data is input by using the text editor "ed." This program is part of the UNIX operating system on the computer rather than the CELDS software. (For questions regarding the use of "ed," the *UNIX Programmer's Manual* should be consulted.) A file should be created and laws typed into this file in sequential order by accession number. Each data field in the file should begin with a "#," followed by a five-digit accession number and a three-digit field number. See the Detailed File Description section in Chapter 5 for a more in-depth discussion of this.

The accession number, field number, and stop code should each be right-justified and zero-filled in the columns indicated. The stop code is 00 if it is not the last line of a field, 01 if it is the last line of a field (but not the last line of the accession number), and 02 if it is the last line of the accession number. For example, the last line of field 7 for accession number 135 begins with a 10-digit code of "0013500701."

When the size of the file approaches 65,000 characters, no more data should be input into it; however, the last line of the file should end with an accession number, i.e., data fields for a single accession number should not be split across file boundaries. At this point, a re-indexing program should be executed for this file by typing "repair <filename>". "Repair" will check field numbers and print out errors; flagged errors should be corrected with "ed." When a file checks out, it should be moved to the /cerl/celds directory and its name changed to "laws.xxxx," where "xxxx" is the accession number of the first law in the file. Data input can now be continued by creating another new file and following the same procedures.

Data Base Creation

The "laws" files that are created as described above comprise the CELDS data files and are read to create the inverted search files which make rapid retrievals possible. The program which creates the search files is "push". Thus, to create a new data base, it is necessary to change the working directory to "/cerl/celds" and execute "push". However, since this is a relatively lengthy process, it is usually run in the background mode, and the output is diverted to a file called "push.out". This is accomplished by typing "push>push.out&". Note that "push" does not modify any of the "laws" files but does read them to create the search files. "Push" must be run to include new laws in the data base or to change the search terms for any laws that have had

searchable fields modified. Merely editing the contents of a "laws" file will change the output that is produced when a particular law is printed; however, changing a "laws" file does not change the search files until "push" is executed.

5 DOCUMENTATION OF CELDS PROGRAM

CELDs Algorithm

CELDs is set up with an inverted index. A group of search files in the inverted index indicates each valid search term and a list of accession numbers which represent "hits" for those terms. Since the length of this list of accession numbers varies, two files are involved; one contains the search term and an address in the second file, and the other contains the list of accession numbers, beginning at the address specified for that search term in the first file. These two files are all that are needed to perform searches; when a search is requested, these files are consulted and the appropriate list of accession numbers is returned to the calling program.

The only remaining important file is a table of contents file, which contains an entry for every accession number in the system. The entry indicates the proper file and the starting address for every field. This file is consulted only when it is necessary to get actual text from a "laws" file.

Thus, the procedure is to use only the inverted search files until the desired law set is established. The table of contents file is then used for each of the accession numbers in the list to locate the desired data fields.

For a rapid identification of search terms, a "hashing" scheme is used to convert textual search terms to a number. Search terms may contain as many as 62 characters (although this is an arbitrary limit). A term is hashed by breaking it into pairs and adding the pairs as integer numbers. For example, "nitrogen dioxide" would be hashed as "ni/tr/og/en d/io/xi/de"; the bit representations for each of the pairs would be treated as if they were integers and added. This would give a large number for the value of "nitrogen dioxide." To fit these values into a table of fixed size, the value is divided by 4001 (this is currently being used as max_hash_num), and the remainder is used to designate the "slot" or "hash_value" of this term. Since the remainder may be any number from 1 to 4000, there are 4000 available slots in the hash table. (Remainder "0" is not used.) However, max_has_num should be picked so that the hash table is never more than approximately two-thirds full; this will insure efficient operations. If a "collision" occurs--that is, two different terms happen to hash to the same value--the next empty slot is used for the second one. This is why the search term itself is also a part of the "val" files (see pp 24,25). The hash number is checked; then terms are compared character by character to guarantee that the term sought and the term found are the same.

Detailed File Description

The files used in CELDS consist of "laws.toc" file, "isol" files, "alpha" files, "val" files, and "list" files.

Laws Files

Laws files contain the data used in CELDS and are selected to be less than 65,536 characters (i.e., 2^{15}) length. The file name is "laws." concatenated with the accession number of the first law in the file; e.g., "laws.131" would be a file beginning with accession number 131. The current laws files are:

laws.1	laws.905	laws.1547	laws.2172	laws.2612
laws.97	laws.931	laws.1573	laws.2185	laws.2616
laws.131	laws.960	laws.1596	laws.2199	laws.2632
laws.165	laws.986	laws.1616	laws.2222	laws.2645
laws.195	laws.1020	laws.1632	laws.2238	laws.2650
laws.224	laws.1043	laws.1650	laws.2259	laws.2663
laws.242	laws.1074	laws.1666	laws.2275	laws.2666
laws.257	laws.1095	laws.1680	laws.2295	laws.2677
laws.282	laws.1120	laws.1700	laws.2306	laws.2692
laws.309	laws.1142	laws.1720	laws.2320	laws.2708
laws.341	laws.1165	laws.1743	laws.2332	laws.2716
laws.373	laws.1193	laws.1772	laws.2352	laws.2731
laws.410	laws.1220	laws.1792	laws.2370	laws.2738
laws.443	laws.1245	laws.1809	laws.2381	laws.2747
laws.481	laws.1268	laws.1825	laws.2393	laws.2760
laws.516	laws.1294	laws.1843	laws.2413	laws.2770
laws.545	laws.1314	laws.1861	laws.2424	laws.2779
laws.579	laws.1327	laws.1884	laws.2445	laws.2793
laws.612	laws.1339	laws.1896	laws.2452	laws.2799
laws.643	laws.1360	laws.1916	laws.2469	laws.2819
laws.679	laws.1376	laws.1937	laws.2482	laws.2831
laws.701	laws.1398	laws.1938	laws.2495	laws.2845
laws.716	laws.1422	laws.1954	laws.2520	laws.2860
laws.737	laws.1440	laws.1967	laws.2536	laws.2883
laws.766	laws.1462	laws.1986	laws.2553	laws.2904
laws.792	laws.1479	laws.2109	laws.2560	laws.2926
laws.823	laws.1499	laws.2122	laws.2573	
laws.856	laws.1521	laws.2141	laws.2587	
laws.883		laws.2158	laws.2595	

The laws are ordered sequentially within the files; therefore, to find any particular accession number (1040, for example), check the list of laws files. In the laws files listed above, there is a file named "laws.1020" and the next one is "laws.1043." The file "laws.1020" will contain accession numbers 1020 to 1042. Therefore, number 1040 must be

in "laws.1020." This information is only necessary for editing, since the CELDS retrieval program will automatically find the appropriate file when CELDS is being used.

Within an accession number, the data fields are in sequential order. Each field must be present and must be preceded by a line beginning with "#," a five-digit accession number, and a three-digit field number. For example, the beginning of field 2 in accession number 5 would be preceded by: #00005002

The following is a sample from the file "laws.2793":

```
#02793001
#02793002 2793
#02793003 transportation of radioactive material.
#02793004 2-10-75
rules and regulations for protection against radiation; part c:
#02793005 licensing of radioactive material; section c.100.
#02793006 transportation
#02793007
dept. of public health
#02793008 535 w. jefferson st., springfield il 62761
#02793009 same as legislative reference
radioactive material shall not be transported outside of the
authorized location of use unless the regulations found in the
following are complied with:
1) 49 cfr, parts 170-189
2) 14 cfr, part 103
3) 46 cfr, part 146
4) 19 cfr, parts 14 and 15
5) illinois vehicle code, chap. 95 1/2, section 12-704.1
these regulations relate to the packaging, marking, storing,
loading, and monitoring of radioactive material, and to the
reporting of accidents.
procedures for opening and closing packages of radioactive
material shall be established and made available to those receiving
the packages
#0273010
#02793011 none
#02793012 damage to vehicles-injuries to humans
radioactive substances
packaging
transportation
```

Laws.toc File

The index to the laws and fields in the "laws" files is kept in a table of contents (toc) file known as the "laws.toc" file. The CELDS retriever uses this file to determine the file that contains a given law

and what character numbers in the file the fields of that law include. The "laws.toc" file contains 13 entries for each accession number. The first entry is the number of the file containing the accession number. For example, a particular law may be in the laws.31 file. For this law, the first entry in "laws.toc" would contain the number 31. The remaining 12 entries are the character numbers within that file which represent the start of each of the 12 additional data fields. Therefore, to locate any law, multiply the accession by 13 and locate that record number in the "laws.toc" file. The first word will indicate which "laws" file to search for the accession number and the next 12 words will indicate the start of each field within that file. The "laws.toc" file is created by the "push" program and is updated or modified by the "repair" program.

Isol Files

The "Isol" files, created by the "push" program, are: agy.isol, mec.isol, gps.isol, att.isol, and top.isol. Each searchable field has an "isol" file created which consists of the searchable fields from the "laws" files--that is, the first line of the agency field, the gps field, the mec field, the attribute field, and the keyword field. A sample from "mec.isol" would look like:

```
00005: air quality
00006: air quality
00039: air quality
00047: air quality
00047: health science
00047: solid waste
00048: air quality
00050: air quality
00051: air quality
00052: air quality
00054: water quality
```

Alpha Files

The "alpha" files are sorted versions of the "isol" files. They are in alphabetical order by search term, and for the same search terms they are in sequential order by accession number. For example, under the key.isol file, "dredging" would be before "estuaries," and under "dredging," the laws would be sequenced by accession number.

Val Files

The "val" file for a field contains all of its search terms and a "hash" table for quick access to those terms. The first part of the file is the hash table. It contains max hash number + 10 "slots" or words. "Max_hash_number" is currently defined to be 4001. The 10

additional slots is merely to allow for the possibility of several different terms hashing to 4000. This number must be greater than the number of searchable terms, since each term must occupy one slot. Each slot contains either a zero, which indicates that no terms hashed to this number, or the record number of the term that hashed to this value. Following the hash table are the records, each of which (one for every term) is 64 characters long. The first 62 characters are the search term itself; the remaining two characters (one word) are an integer number that gives the position in the "list" file of the beginning of the list of accession numbers associated with this term. Therefore, to locate the list of laws associated with a search term, determine the hash number and look in that "slot" in the hash table. This will give the record number in the val file. Calculate the character number with which the record begins by multiplying the record number by 64 (the number of characters per record) and adding the size in characters of the hash tables (since this precedes the records in the val file).

The "val" files are created by the "make_search" program and are used in the CELDS retriever to do searches.

List Files

The "list" files are lists of law numbers terminated by 19999 and are of variable length. The law numbers are stored as integers, i.e., one word (two characters) per number. The entry contained in the "val" file is the position of the start of a law list, i.e., its number in the "list" file. To convert this to the starting character number of a law list, multiply by two, since there are two characters per integer number.

Documentation of Programs

Repair

"Repair" is the program which reads a "laws" file and prepares the "laws.toc" file. A variable keeps the current character number as the file is read. When a new field is encountered, the appropriate address is entered into the "laws.toc" file.

Some data validation is also performed by "repair." Accession numbers and field numbers are checked for sequential order. Lines containing errors are printed. If "repair" is executed without a "_" argument, validation is the only task performed.

If "repair" is executed with a "_" argument (e.g., "repair - laws.1"), "isol" files are produced (see Detailed File Description section). As searchable fields are encountered in the input stream, they are copied to an appropriate "isol" file (e.g., "key.isol" for key-

words). In standard operation "repair" is executed with a " " argument only by the "push" program. Appendix C provides a documented copy of the source code.

Make_Search

"Make_search" is the program which reads a sorted version of the "isol" files produced by "repair" and creates "val" and "list" files for each searchable field. It requires an argument indicating which searchable field is being prepared. For example, "make_search mec" will use "mec.alpha" to produce the files "mec.list" and "mec.val." Appendix D provides a documented copy of this source code.

Hier

"Hier" reads a static thesaurus file ("key.hier") and creates hierarchical search files ("list" and "val"). Currently, this applies only to the keyword field. Searches in the keyword field are presumed to be hierarchical, i.e., a search for a broad term will also include all narrower terms under it. The nonhierarchical file is referred to as "top" (for topic), while the hierarchical file is "key" (e.g., "top.val," "key.val," etc.).

"Top.val" and "top.list" files are produced by "make_search top." These two files are then used by "hier" to construct lists of laws for the terms in the thesaurus file. Appendix E provides the documented source.

Push

"Push" is the shell program (command file) which is executed to create a new data base. It systematically executes "repair" for every "laws" file, and then sorts the "isol" files produced into the "alpha" files required by the "make_search" program. Next, it executes "make_search" for each of the CELDS searchable fields. Finally, "hier" is executed to produce the hierarchical keyword file.

"Push" is a shell file. The procedures it uses include "repair" and "make_search." Appendix F provides a documented copy of this file.

CELDS

The "CELDS" program, sometimes referred to as the retriever, is the main part of the CELDS system. It provides the user interface and performs the requested searching and listing of information. A few conventions involving global variables are used throughout the CELDS routines. An input line is read into a buffer called "request." Two pointers into this buffer are maintained: (1) "old_request_location" points to the previous position in the line, and (2) request_location

points to the current position in the request line. The word that is currently being processed in the request line is stored in the array "word" and is null-terminated. Every routine that uses a word gets the next word and places it into the "word" array in preparation for the next routine to be called. Thus, every routine expects that "word" is already prepared for it; in turn, it fixes "word" for the next routine. If the line terminates, then a null is placed into word[0]. In all cases, "get next word" is called to provide the next word from a line. Appendix G provides the documented source code for the retriever routines.

Library Routines

Besides the routines appearing in the appendices, several routines of general utility to the CELDS programs are kept in a library. These include routines of the type to do hashing, input/output, concatenation of strings, etc. Appendix H provides these routines. One other file also included in Appendix H is "search.i." This is an "include" file (see *C Reference Manual*⁸). It contains constants used by most of the CELDS programs, such as hash table size, maximum number of laws, etc. Each program that depends on these has an "include" statement which has the effect of incorporating the "search.i" file into the source code. Changing a parameter in the "search.i" file will therefore cause it to be changed in all of the CELDS programs, thus eliminating errors due to oversight.

⁸ Dennis M. Ritchie, *C Reference Manual* (Bell Telephone Laboratories).

6 SUMMARY AND RECOMMENDATION

CELDS contains abstracts of environmental legislation for the Federal government and for all 50 states and Puerto Rico. This report has provided complete documentation of CELDS, including background information, description of data records, how the information is abstracted, indexed, and updated, and listings of the software. The documentation described in this report should be used for any future modification, update, and maintenance of CELDS.

CELDS should be brought up in an operational environment and made available to all elements of the Army to aid with environmental questions.

Appendix A - Attribute Listing

AIR QUALITY

ENV INFLUENCE FAC

AIR MASS

STABILITY
TEMPERATURE
MIXING DEPTH
WIND SPEED
WIND DIRECTION
HUMIDITY
PRECIPITATION

LAND MASS

ALBEDO
INSOLATION
TOPOGRAPHY

PARTICULATES

AGGREGATE

DUST AND FUMES
FLY ASH
SMOKE AND SOOT

INORG SOLIDS, MISTS

ALUMINUM AND COMPOUNDS
ARSENIC AND COMPOUNDS
ASBESTOS
BARIUM AND COMPOUNDS
BERYLLIUM AND COMPOUNDS
BORON AND COMPOUNDS
CADMIUM AND COMPOUNDS
CALCIUM AND COMPOUNDS
CHROMIUM AND COMPOUNDS
COPPER AND COMPOUNDS
IRON AND COMPOUNDS
LEAD AND COMPOUNDS
MANGANESE AND COMPOUNDS
MOLYBDENUM AND COMPOUNDS
NICKEL AND COMPOUNDS
SELENIUM AND COMPOUNDS
SILICON AND COMPOUNDS
SILVER AND COMPOUNDS
SODIUM AND COMPOUNDS
THALLIUM AND COMPOUNDS
TIN AND COMPOUNDS
TITANIUM AND COMPOUNDS
TUNGSTEN AND COMPOUNDS
VANADIUM AND COMPOUNDS
ZINC AND COMPOUNDS
ZIRCONIUM AND COMPOUNDS
RADIOACTIVE SUBSTANCES
FLUORINE AND COMPOUNDS
SULFUR AND COMPOUNDS

CHLORINE AND COMPOUNDS
BROMINE AND COMPOUNDS
IODINE AND COMPOUNDS
PHOSPHOROUS AND COMPOUNDS
MERCURY AND COMPOUNDS
NITROGEN AND COMPOUNDS
MAGNESIUM AND COMPOUNDS
POTASSIUM AND COMPOUNDS
ANTIMONY AND COMPOUNDS
ORGANIC COMPOUNDS
SATURATED HYDROCARBONS
CYCLIC SATURATED HYDROCARBONS
UNSATURATED HYDROCARBONS
AROMATIC HYDROCARBONS
ALCOHOLS
PHENOLS
ETHERS
AMINES
ALDEHYDES
KETONES
ORGANIC ACIDS AND DERIVATIVES
ORGANIC SULFUR
ORGANIC HALIDES
BIOLOGICAL
AEROALLERGENS
ALLERGENS (EXCLUDING AEROALLERGENS)
FUNGI
BACTERIA
VIRUSES
PARTICULATE BIOCIDES
INSECTICIDES
MITICIDES AND NEMATOCIDES
RODENTICIDES AND FUNGICIDES
HERBICIDES
GASES AND VAPORS
INORGANIC
SULFUR AND COMPOUNDS
NITROGEN AND COMPOUNDS
BROMINE AND COMPOUNDS
OZONE
CHLORINE AND COMPOUNDS
FLUORINE AND COMPOUNDS
RADIOACTIVE
ORGANIC
SATURATED HYDROCARBONS
CYCLIC SATURATED HYDROCARBONS
UNSATURATED HYDROCARBONS
AROMATIC HYDROCARBONS
ALCOHOLS

PHENOLS
ETHERS
AMINES
ALDEHYDES
KETONES
ORGANIC ACIDS AND DERIVATIVES
SULFUR
HALIDES
RADIOACTIVE
CARBON AND COMPOUNDS
GASEOUS BIOCIDES
INSECTICIDES
MITICIDES AND NEMATOCIDES
RODENTICIDES AND FUNGICIDES
HERBICIDES

CNTRVSL

PARTICULATE MATTER
SULFUR OXIDES
HYDROCARBONS
PHOTOCHEMICAL OXIDANTS
CARBON MONOXIDE
OXIDES OF NITROGEN
ODORS
RADIOACTIVE EMISSIONS
AESTHETIC CONSIDERATIONS

EARTH SCIENCE

SITE ATT

TOPOGRAPHY

SLOPE

SUBSTRATUM

HYDROLOGIC REGIME

PRECIPITATION

BEDROCK

PROCESS AT

SUBSTRATUM

SOIL COMPACTION

SOIL HORIZON MIXING

SUBSURFACE VIBRATION

EROSION + TRANSPORT

WATER EROSION

ICE EROSION

WIND EROSION

GRAVITY, MASS WASTING

CNTRVSL

WATER EROSION

HYDROLOGIC REGIME

SUBSURFACE VIBRATION

WIND EROSION

GRAVITY, MASS WASTING

LANDSCAPE AESTHETICS

ECOLOGY

ECOSYSTEM

KINDS OF ANIMALS

- LARGE MAMMALS
- SMALL MAMMALS
- BIRDS
- FISH
- AMPHIBIANS
- INSECTS
- OTHER ANIMALS
- ENDANGERED ANIMAL SPECIES

KINDS OF PLANTS

- TREES
- SHRUBS
- HERBS
- ALGAE
- FUNGI
- lichens
- OTHER PLANT SPECIES
- ENDANGERED PLANT SPECIES

SYSTEM STABILITY

- FOOD WEBS
- PRODUCTIVITY
- SEASONAL ASPECT
- STRATIFICATION
- SUCCESSIONAL STAGE

WILDLIFE MANAGEMENT

HUNTING

- SMALL GAME HUNTING
- WATERFOWL HUNTING
- BIG GAME HUNTING

FISHING

- BOTTOM LIFE
- WARM WATER FISHING
- COLD WATER FISHING
- LARGE LAKE FISHING
- COASTAL WATER FISHING
- SHELLFISH
- DEEP SEA FISHING

PESTS

- DISEASE VECTORS
- NOXIOUS WEEDS
- OTHER UNDESIRABLE SPECIES

CNTRVSL

- IMPACTS ON GAME ANIMALS
- ENCROACHMENT ON NATURAL HABITATS
- THREATENED SPECIES

HEALTH SCIENCE

BIOLOGICAL

POLLEN
VIRUS
RICKETTSIA
PROTOZOA
BACTERIA
FUNGI
WORMS
ARTHROPODS
RODENTS

CHEMICAL

CARBON MONOXIDE
SULFUR DIOXIDE
NITROGEN AND NITROGEN OXIDES
PARTICULATE MATTER
LEAD
MERCURY
ACIDS
CADMIUM
ARSENIC
SELENIUM
PESTICIDES AND RESIDUES
BARIUM
CHROMIUM
COPPER
NICKEL
ZINC
DETERGENTS
HALOGENS
SULFUR
PHENOLS
CYANIDE
METHANE
CARCINOGENIC SUBSTANCES
ALUMINUM
BERYLLIUM
SILICON
THALLIUM
ASBESTOS
ALCOHOLS
ALDEHYDES
KETONES
ETHERS

PSYCHOLOGICAL

MIL + CIV ARMY PERSONNEL
WORK OVEREXPOSURE
INADEQUATE TRAINING

DISLOCATION ADJUSTMENTS
 ARMY DISCIPLINE
 PERSONNEL POLICIES
 PHYSICAL OVEREXPOSURE
 ECONOMIC HARDSHIPS
 INDIV IN COMMUNITY NEAR INST
 MILITARY SECRECY
 VISUAL ENVIRONMENTAL CHANGES
 COMMUNICATIONS NETWORK INTERFERENCE
 BOTH ARMY PERSONNEL + PRIVATE INDIV
 TRAFFIC OVEREXPOSURE
 TRAUMATIC EXPERIENCES
 POLLUTANT OVEREXPOSURE
 HOUSING CONDITIONS
 POPULATION CHANGE

SAFETY

TRANSPORTATION SAFETY
 AIR
 GROUND
 WATER
 RESIDENTIAL OR HOME AREA
 COMMUNITY/MARKETING
 WORK
 RECREATION
 RADIATIONS
 RADIATION-IONIZING
 RADIATION-MICROWAVE
 RADIATION-LASER
 OTHER RADIATION

CNTRVSL

EXPOSURE TO CARCINOGENS/MUTAGENS
 HARMFUL FOODS/WATER ADDITIVES
 PSYCHOLOGICAL STRESSORS
 DRUG + NARCOTICS ABUSE
 ENDANGERING COMMUNITY HEALTH
 ENDANGERING COMMUNITY SAFETY

LAND USE

CONSUMPTION

CONSUMPTION OF LAND

CONFLICT

ACCESS TO MINERALS

INTERFERENCE OFF OF POST

INCOMPATIBILITY ON POST

CHANGE

INDUCED LAND-USE CHANGES

CNTRVSL

CONSUMPTION OF LAND

ACCESS TO MINERALS

INTERFERENCE OFF OF POST

INDUCED LAND-USE CHANGES

NOISE

PHYSIOLOGICAL MAINTENANCE
SLEEP PERFORMANCE
TASK PERFORMANCE
AURAL COMMUNICATION
TELEVISION/RADIO COMMUNICATION
LAND USE INCOMPATIBILITY AND INTEGRITY
CNTRVSL

COMMUNITY ANNOYANCE
PROPERTY VALUE DEPRECIATION

SOCIOLOGY

HUMAN ECOLG

POPULATION

SIZE
COMPOSITION
NET CHANGE

HUMAN ECOLG

RURAL AREAS
URBAN AREAS
SUBURBS
URBAN FRINGE

SOC STRUCT

SOCIAL CATEGORIES

AGE CATEGORIES
SEX CATEGORIES
FAMILY STATUS CATEGORIES

SOCIAL CLASSES

UPPER CLASS
MIDDLE CLASS
LOWER CLASS

ASSOCIATIONS

VOLUNTARY ASSOCIATION
ORGANIZATIONS

INSTITUTIONS

FAMILIES
EDUCATIONAL ORGANIZATIONS
RELIGIOUS ORGANIZATIONS

SOCIAL CONTROL

LAW ENFORCEMENT

SOCL PROC

SOCIAL CONTROL

COURTS
POLITICAL PROCESS
WELFARE AND DEPENDENCY

PUBLIC OPINION

PUBLICS
OPINION LEADERS
OPINION PROCESS

MASS COMMUNICATIONS

PRINTED MEDIA
BROADCAST MEDIA

CNTRVSL

POPULATION
ECOLOGY
EDUCATIONAL ORGANIZATIONS
SOCIAL CONTROL
PUBLIC OPINION
MASS COMMUNICATION

AESTHETIC CHARACTER OF COMMUNITY

SOLID WASTE

COLLECTION
DISPOSAL
MANAGEMENT

TRANSPORTATION

ROAD TRANS

DISRUPTIONS IN HIGHWAY TRAFFIC FLOW
INDUCED MODIFICATION TO HIGHWAYS
POLLUTION FROM HIGHWAYS
DAMAGE TO HIGHWAYS
DAMAGE TO VEHICLES -INJURIES TO HUMANS

RAIL TRANS

DISRUPTION TO RAILWAY TRAFFIC
INDUCED MODIFICATION TO RAILWAYS
POLLUTION FROM RAILWAYS
DAMAGE TO RAILWAYS

AIR TRANS

DISRUPTION TO AIRFIELD TRAFFIC
INDUCED MODIFICATION TO AIRFIELDS
POLLUTION FROM AIRFIELDS
DAMAGE TO AIRFIELDS

WATER TRAN

DISRUPTION TO WATERWAY TRAFFIC
INDUCED MODIFICATION TO WATERWAYS
POLLUTION FROM WATERWAYS
DAMAGE TO WATERWAYS

CNTRVSL

DISRUPTIONS IN HIGHWAY TRAFFIC FLOWS
DAMAGE TO VEHICLES-INJURIES TO HUMANS
INDUCED MODIFICATION TO HIGHWAYS
INDUCED MODIFICATION TO AIRFIELDS

WATER QUALITY

PHYS ENVMT

AQUIFER CHAR

AVAILABILITY OF GROUND WATER

WATER QUALITY PARAMS

TURBIDITY

TEMPERATURE

COLOR

SUSPENDED SOLIDS

GROSS SOLIDS

SETTLEABLE SOLIDS

FLOATING SOLIDS

VOLATILE SUSPENDED SOLIDS

TASTE AND ODOR

OILS

DISSOLVED GASES

STREAM OR WATER BODY

DEPTH

VELOCITY

SOLAR RADIATION INTENSITY

WIND VELOCITY AND DIRECTION

DYNAMIC PRESSURE

ATMOSPHERIC REAERATION

MORPHOMETRY AND FLOW PATTERN

SUBSTRATUM

DEPENDABLE YIELD

MAXIMUM DISCHARGE

MINIMUM DISCHARGE

RATE OF CHANGE OF DISCHARGE

CHEM ENVMT

INORGANIC

IRON

MANGANESE

SODIUM

CALCIUM

MAGNESIUM

NITROGEN

PHOSPHORUS

ARSENIC

BARIUM

BORON

CADMIUM

CHROMIUM

COPPER

FLUORIDE

LEAD

MERCURY

NICKEL

SELENIUM
SILVER
ZINC
ALKALINITY AND ACIDITY
HYDROGEN ION CONCENTRATION (PH)
OXIDATION REDUCTION POTENTIAL (EH)
DISSOLVED CARBON DIOXIDE
TOTAL DISSOLVED SOLIDS
CHLORIDE
SULFUR
DISSOLVED OXYGEN
SALINITY
OTHER INORGANIC CHEMICALS
ORGANIC
BOD
COD
PHENOLS
DETERGENTS
CARCINOGENIC SUBSTANCES
CARBON CHLOROFORM EXTRACT (CCE)
CYANIDE
METHANE
OTHER ORGANIC COMPOUNDS
BIOCIDES
PESTICIDES
RADIOACTIVE
RADIOACTIVITY
BIOLOGICAL
PATHOGENIC
PATHOGENIC VIRUSES
PATHOGENIC BACTERIA
PATHOGENIC PROTOZOA
OTHER PATHOGENIC ORGANISMS
AQUATIC LIFE
PLANKTON
BENTHOS
NEKTON
OTHER ORGANISMS
COMMUNITY MAINTENANCE
CNTRVSL
SYNTHETIC DETERGENTS
FLUORIDATION
WATER QUANTITY
MERCURY
OIL
THERMAL POLLUTION
OTHER POTENTIALLY CONTROVERSIAL ASPECTS
AQUIFER YIELD

Appendix B - Keyword Listing

ACCIDENTS

SN UNINTENTIONAL RELEASES OF CONTAMINANTS INTO THE
AIR OR WATER.
NT OIL SPILLS

ACIDS

BT INORGANIC COMPOUNDS
NT NITRIC ACID
SULFURIC ACID
RT HAZARDOUS MATERIALS
*

AGRICULTURAL POLLUTION

NT FEEDLOTS
GRAIN HANDLING
RT COTTON GINS
EROSION
FERTILIZERS
HERBICIDES
PESTICIDES
RENDERING

AIR POLLUTION CONTROL

SN DEVICE OR PROCEDURE USED TO LIMIT THE RELEASE
OF CONTAMINANTS INTO THE AIR.

AIR POLLUTION EPISODES

SN STATUS DECLARED BY STATE OFFICIALS WHEN AIR
CONTAMINANTS REACH HIGH LEVELS; EMISSION REDUCTION
PLANS MUST THEN BE ADHERED TO.

AIR POLLUTION SOURCES

NT ASPHALT PLANTS
BOILERS
CEMENT PLANTS
COATINGS
COKE OVENS
COTTON GINS
FERROALLOYS

STEEL

FOUNDRIES
FURNACES

BLAST FURNACES
CUPOLAS

GRAIN HANDLING
HEAT EXCHANGERS
INCINERATORS
CONICAL BURNERS
INDIRECT SOURCES
AIRPORTS
ROADS

INDUSTRIAL COOLING
LANDFILLS
 SANITARY LANDFILL
MANUFACTURING
 CHEMICAL MANUFACTURING
NONFERROUS METALS
 ARSENIC
 BARIUM
 BERYLLIUM
 CADMIUM
 CHROMIUM
 COPPER
 LEAD
 MANGANESE
 MERCURY
 NICKEL
 SILVER
 SODIUM
 ZINC
OPEN BURNING
POWER SOURCES
 INTERNAL COMBUSTION ENGINES
 DIESEL ENGINES
 GASOLINE ENGINES
 NUCLEAR ENERGY
 STEAM GENERATING PLANTS
 TURBINES
PULP MILLS
SEPARATION PROCESSES
SINTERING
SMELTERS
SPRAYING
STOCKPILES
VEHICLES
AIR QUALITY CLASSIFICATION
 BT CLASSIFICATION
 RT LAND CLASSIFICATION
 WATER CLASSIFICATION
AIR QUALITY CONTROL REGIONS
 USE AQCR, SPECIFIC
AIR QUALITY STANDARDS
 RT EMISSION STANDARDS
AIRBORNE PARTICULATES
 UF PARTICULATES
 NT ASH
 DUST

		FUMES
		MISTS
		SMOKE
RT		OPACITY
AIRCRAFT		
RT		VEHICLES
		WATERCRAFT
AIRPORTS		
BT		AIR POLLUTION SOURCES
		INDIRECT SOURCES
RT		ROADS
ALCOHOLS		
BT		ORGANIC COMPOUNDS
RT		*
ALDEHYDES		
BT		ORGANIC COMPOUNDS
RT		*
ALKYL BENZENE SULFONATES		
BT		INORGANIC COMPOUNDS
		SULFUR
RT		SULFUR OXIDES
		SULFURIC ACID
AMMONIA		
BT		INORGANIC COMPOUNDS
NT		AMMONIA NITROGEN
RT		HAZARDOUS MATERIALS
		*
AMMONIA NITROGEN		
BT		INORGANIC COMPOUNDS
		AMMONIA
AMMUNITION		
BT		EXPLOSIVES
RT		*
AQCR, SPECIFIC		
SN		A COLLECTIVE KEYWORD FOR SPECIFIC AQCR'S WHICH
		HAVE BEEN TREATED INDIVIDUALLY IN THE REGULATIONS
		AND DATA BASE; NAMES OF AQCR'S ARE NOT LISTED
		IN THE THESAURUS.
UF		AIR QUALITY CONTROL REGIONS

AQUATIC ANIMALS

BT AQUATIC LIFE
RT AQUATIC PLANTS
FISH

AQUATIC LIFE

NT AQUATIC ANIMALS
AQUATIC PLANTS
FISH
RT FLORA
PROTECTED SPECIES
WILDLIFE

AQUATIC PLANTS

BT AQUATIC LIFE
RT AQUATIC ANIMALS
FISH

ARSENIC

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

ASBESTOS

BT INORGANIC COMPOUNDS
SILICATES
POINT SOURCES
RT FELDSPARS
HAZARDOUS MATERIALS
*

ASH

BT AIRBORNE PARTICULATES
RT DUST
FUMES
MISTS
SMOKE

ASPHALT PLANTS

BT AIR POLLUTION SOURCES
RT *

ATLANTIC OCEAN

RT COASTS
SALINE WATER

WATERWAYS
WETLANDS

BACTERIA
NT FECAL COLIFORMS
RT HAZARDOUS MATERIALS

BARIUM
BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

BASINS
USE BAYS, SPECIFIC

BAYS, SPECIFIC
SN A COLLECTIVE KEYWORD FOR SPECIFIC BAYS WHICH HAVE
BEEN TREATED INDIVIDUALLY IN THE REGULATIONS
AND DATA BASE; NAMES OF BAYS ARE NOT LISTED
IN THE THESAURUS.
UF BASINS
HARBORS
RT SEAPORTS

BERYLLIUM
BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

BIOCHEMICAL OXYGEN DEMAND
USE BOD

BIOLOGICAL WARFARE AGENTS
RT CHEMICAL WARFARE AGENTS
HAZARDOUS MATERIALS

BLACK POWDER
BT EXPLOSIVES
RT *

BLAST FURNACES
BT AIR POLLUTION SOURCES

		FURNACES
		POINT SOURCES
		FURNACES
	RT	CUPOLAS
BLASTING CAPS		
	BT	EXPLOSIVES
	RT	*
BOD		
	UF	BIOCHEMICAL OXYGEN DEMAND
	RT	COD
		DISSOLVED OXYGEN
BOILERS		
	BT	AIR POLLUTION SOURCES
		POINT SOURCES
	RT	*
BORON		
	BT	INORGANIC COMPOUNDS
	RT	*
CADMIUM		
	BT	AIR POLLUTION SOURCES
		NONFERROUS METALS
		INORGANIC COMPOUNDS
		NONFERROUS METALS
		POINT SOURCES
		NONFERROUS METALS
	RT	*
CANNON AMMUNITION		
	BT	EXPLOSIVES
	RT	*
CARBON		
	BT	ORGANIC COMPOUNDS
	NT	CARBON MONOXIDE
		ORGANIC CARBON
	RT	CCE
		HYDROCARBONS
		*
CARBON CHLOROFORM EXTRACT		
		USE CCE
CARBON MONOXIDE		
	BT	ORGANIC COMPOUNDS

		CARBON
	RT	ORGANIC CARBON OXIDANTS
CCE		
	UF	CARBON CHLOROFORM EXTRACT
	BT	ORGANIC COMPOUNDS
	RT	CARBON *
CEMENT PLANTS		
	BT	AIR POLLUTION SOURCES POINT SOURCES
	RT	*
CHANNELIZATION		
	SN	ANY ACT WHICH AFFECTS THE BED OR ROUTE OF A BODY OF WATER.
	NT	DREDGING
CHANNELS		
	RT	WATERWAYS
CHEMICAL AMMUNITION		
	BT	EXPLOSIVES
	RT	*
CHEMICAL MANUFACTURING		
	SN	TERM TO DENOTE POINT SOURCES WHICH MANUFACTURE INORGANIC OR ORGANIC CHEMICALS
	BT	AIR POLLUTION SOURCES MANUFACTURING
	RT	POINT SOURCES INORGANIC COMPOUNDS ORGANIC COMPOUNDS *
CHEMICAL OXYGEN DEMAND		
		USE COD
CHEMICAL WARFARE AGENTS		
	RT	BIOLOGICAL WARFARE AGENTS HAZARDOUS MATERIALS
CHLORIDES		
	NT	VINYL CHLORIDES
	RT	HAZARDOUS MATERIALS

CHLORINE

BT INORGANIC COMPOUNDS
RT *

CHROMIUM

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

CITIES

USE URBAN AREAS
URBAN AREAS, SPECIFIC

CLASSIFICATION

SN QUALITY AND/OR USE CLASSIFICATION FOR LAND OR
WATER; ADMINISTRATIVE REGIONS FOR AIR QUALITY.
NT AIR QUALITY CLASSIFICATION
LAND CLASSIFICATION
WATER QUALITY CLASSIFICATION

COAL

BT FUELS
RT COKE
LIQUID FUELS
PETROLEUM
WOOD

COASTS

RT ATLANTIC OCEAN
PACIFIC OCEAN
SALINE WATER
SEAPORTS
TIDAL WATER
WETLANDS

COATINGS

SN SUBSTANCES APPLIED TO SURFACES BY ELECTROPLATING
OR SPRAYING IN A MANNER PERMITTING RELEASE OF
POLLUTANTS; E.G., PAINTS OR METALS.
UF ELECTROPLATING
BT AIR POLLUTION SOURCES
POINT SOURCES
RT HAZARDOUS MATERIALS
SPRAYING
*

COD

UF CHEMICAL OXYGEN DEMAND
RT BOD
DISSOLVED OXYGEN

COKE

BT FUELS
RT COAL
LIQUID FUELS
PETROLEUM
WOOD

COKE OVENS

BT AIR POLLUTION SOURCES
POINT SOURCES
RT *

COLIFORM BACTERIA

USE FECAL COLIFORMS

COLOR

CONDUCTIVITY

CONICAL BURNERS

BT AIR POLLUTION SOURCES
INCINERATORS
WASTE DISPOSAL
INCINERATORS

CONTAINERS

RT PACKAGING
STORAGE TANKS

COPPER

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS

RT *

COTTON GINS

BT AIR POLLUTION SOURCES
RT AGRICULTURAL POLLUTION
*

COUNTIES, SPECIFIC

SN A COLLECTIVE KEYWORD FOR SPECIFIC COUNTIES WHICH
HAVE BEEN TREATED INDIVIDUALLY IN THE REGULATIONS
AND DATA BASE; NAMES OF COUNTIES ARE NOT LISTED
IN THE THESAURUS.

CRUDE OIL

USE PETROLEUM

CUPOLAS

BT AIR POLLUTION SOURCES
FURNACES
POINT SOURCES
FURNACES
RT BLAST FURNACES

CYANIDES

BT INORGANIC COMPOUNDS
RT HAZARDOUS MATERIALS
*

DEPOSITION

RT EROSION
SEDIMENTATION
SETTLEABLE SOLIDS

DESIGN CRITERIA

DETONATING DEVICES

BT EXPLOSIVES
RT *

DIESEL ENGINES

BT AIR POLLUTION SOURCES
POWER SOURCES
INTERNAL COMBUSTION ENGINES
POINT SOURCES
POWER SOURCES
INTERNAL COMBUSTION ENGINES
RT GASOLINE ENGINES

DISPERSANTS

UF EMULSIFIERS
RT OIL SPILLS
SOLVENTS

DISSOLVED OXYGEN

RT BOD
COD

DISSOLVED SOLIDS

RT SETTLEABLE SOLIDS
SUSPENDED SOLIDS

DREDGING

BT CHANNELIZATION

DRINKING WATER

USE POTABLE WATER

DUMPING GROUNDS

SN SOLID WASTE DISPOSAL AREAS IN A BODY OF WATER.
BT WASTE DISPOSAL
RT GARBAGE COLLECTION
INCINERATORS
JUNKYARDS
LANDFILLS
OPEN BURNING
OPEN DUMPING
TRANSFER STATIONS
WASTE PROCESSING

DUST

BT AIRBORNE PARTICULATES
RT ASH
FUMES
MISTS
SMOKE

ECONOMIC POISONS

USE HERBICIDES
PESTICIDES

EFFLUENT STANDARDS

RT WATER QUALITY STANDARDS

EFFLUENTS

NT INDUSTRIAL WASTES
PROCESS WASTE WATER
SEWAGE
RT MIXING ZONE
POINT SOURCES
THERMAL POLLUTION

ELECTROPLATING

USE COATINGS

EMISSION STANDARDS

RT	AIR QUALITY STANDARD
EMISSIONS	
NT	EXHAUST EMISSIONS
EMULSIFIERS	
	USE DISPERSANTS
ENDANGERED SPECIES	
BT	PROTECTED SPECIES
RT	THREATENED SPECIES
EROSION	
RT	AGRICULTURAL POLLUTION DEPOSITION SEDIMENTATION SETTLEABLE SOLIDS
ESTUARIES	
BT	TIDAL WATER
ETHYLENE	
BT	ORGANIC COMPOUNDS
RT	*
EXHAUST EMISSIONS	
BT	EMISSIONS
EXHAUST SYSTEMS	
SN	TERM INCLUDES EXHAUST AND VENTILATING SYSTEMS.
EXPLOSIVE BOMBS	
BT	EXPLOSIVES
RT	*
EXPLOSIVE GRENADES	
BT	EXPLOSIVES
RT	*
EXPLOSIVE MINES	
BT	EXPLOSIVES
RT	*
EXPLOSIVE POWER DEVICES	
BT	EXPLOSIVES
RT	*
EXPLOSIVE PROJECTILES	
BT	EXPLOSIVES

RT	*
EXPLOSIVE TORPEDOES	
BT	EXPLOSIVES
RT	*
EXPLOSIVES	
NT	AMMUNITION
	BLACK POWDER
	BLASTING CAPS
	CANNON AMMUNITION
	CHEMICAL AMMUNITION
	DETONATING DEVICES
	EXPLOSIVE BOMBS
	EXPLOSIVE GRENADES
	EXPLOSIVE MINES
	EXPLOSIVE POWER DEVICES
	EXPLOSIVE PROJECTILES
	EXPLOSIVE TORPEDOES
	GAS MINES
	GAS PROJECTILES
	HIGH EXPLOSIVES
	IGNITERS
	INCENDIARY PROJECTILES
	INITIATING EXPLOSIVES
	JET THRUST UNITS
	LOW EXPLOSIVES
	NONEXPLOSIVE AMMUNITION
	PROPELLANT EXPLOSIVES
	ROCKET AMMUNITION
	ROCKET MOTORS
	STARTER CARTRIDGES
RT	HAZARDOUS MATERIALS
	PACKAGING
	STORAGE
	TRANSPORTATION
FECAL COLIFORMS	
UF	COLIFORM BACTERIA
BT	BACTERIA
FEEDLOTS	
BT	ARGICULTURAL POLLUTION
	POINT SOURCES
RT	GRAIN HANDLING
	*
FELDSPARS	
BT	INORGANIC COMPOUNDS

		SILICATES
	RT	ASBESTOS
FERROALLOYS		
	BT	AIR POLLUTION SOURCES
		POINT SOURCES
	NT	STEEL
	RT	IRON
		SMELTERS
		*
FERTILIZERS		
	BT	POINT SOURCES
	RT	AGRICULTURAL POLLUTION
		*
FIRES		
	RT	OPEN BURNING
FISH		
	SN	TERM INCLUDES SHELLFISH; DISTINGUISHED FROM OTHER
		AQUATIC ANIMALS MAINLY BY ECONOMIC IMPORTANCE.
	UF	SHELLFISH
	BT	AQUATIC LIFE
	RT	AQUATIC ANIMALS
		AQUATIC PLANTS
FLOATING DEBRIS		
FLOOD CONTROL		
FLORA		
	UF	PLANT LIFE
	RT	AQUATIC LIFE
		PROTECTED SPECIES
		WILDLIFE
FLUORIDES		
	BT	INORGANIC COMPOUNDS
	RT	*
FOREST PRESERVATION		
	RT	LAND PRESERVATION
FOUNDRIES		
	BT	AIR POLLUTION SOURCES
	RT	*
FUEL OIL		

	BT	FUELS
		LIQUID FUELS
	RT	GASOLINE
FUELS		
	NT	COAL
		COKE
		LIQUID FUELS
		FUEL OIL
		GASOLINE
		WOOD
FUMES		
	BT	AIRBORNE PARTICULATES
	RT	ASH
		DUST
		MISTS
		SMOKE
FURNACES		
	BT	AIR POLLUTION SOURCES
		POINT SOURCES
	NT	BLAST FURNACES
		CUPOLAS
	RT	*
GARBAGE COLLECTION		
	BT	WASTE DISPOSAL
	RT	DUMPING GROUNDS
		INCINERATORS
		JUNKYARDS
		LANDFILLS
		OPEN BURNING
		OPEN DUMPING
		TRANSFER STATIONS
		WASTE PROCESSING
GAS MINES		
	BT	EXPLOSIVES
	RT	*
GAS PROJECTILES		
	BT	EXPLOSIVES
	RT	*
GASOLINE		
	BT	FUELS
		LIQUID FUELS
	RT	FUEL OIL

GASOLINE ENGINES

BT AIR POLLUTION SOURCES
POWER SOURCES
INTERNAL COMBUSTION ENGINES
SOURCES
POINT SOURCES
POWER SOURCES
INTERNAL COMBUSTION ENGINES
RT DIESEL ENGINES

GRAIN HANDLING

BT AGRICULTURAL POLLUTION
AIR POLLUTION SOURCES
POINT SOURCES
RT FEEDLOTS
*

HARBORS

USE BAYS, SPECIFIC

HAZARDOUS MATERIALS

RT ACIDS
AMMONIA
ASBESTOS
BACTERIA
BIOLOGICAL WARFARE AGENTS
CHEMICAL WARFARE AGENTS
CHLORIDES
COATINGS
CYANIDES
EXPLOSIVES
HERBICIDES
INDUSTRIAL WASTES
NONFERROUS METALS
OILS
PESTICIDES
RADIOACTIVE SUBSTANCES
SEWAGE
SLUDGE
SOLVENTS
TOXIC SUBSTANCES
VOLATILE SUBSTANCES

HEAT EXCHANGERS

UF INDIRECT HEAT EXCHANGERS
BT AIR POLLUTION SOURCES
RT *

HERBICIDES

UF ECONOMIC POISONS
BT PESTS
RT AGRICULTURAL POLLUTION
HAZARDOUS MATERIALS
PEST CONTROL
PESTICIDES

HIGH EXPLOSIVES

BT EXPLOSIVES
RT *

HYDROCARBONS

BT ORGANIC COMPOUNDS
RT CARBON
HYDROGEN
*

HYDROGEN

BT INORGANIC COMPOUNDS
NT HYDROGEN FLUORIDE
HYDROGEN SULFIDE
RT HYDROCARBONS
PH
*

HYDROGEN FLUORIDE

BT INORGANIC COMPOUNDS
HYDROGEN
RT HYDROGEN SULFIDE

HYDROGEN ION CONCENTRATION
USE PH

HYDROGEN SULFIDE

BT INORGANIC COMPOUNDS
HYDROGEN
RT HYDROGEN FLUORIDE

IGNITERS

BT EXPLOSIVES
RT *

IMPOUNDMENTS OF WATER
UF RESERVOIRS

INCENDIARY PROJECTILES

BT EXPLOSIVES
RT *

INCINERATORS

BT AIR POLLUTION SOURCES
WASTE DISPOSAL
NT CONICAL BURNERS
RT DUMPING GROUNDS
GARBAGE COLLECTION
JUNKYARDS
LANDFILLS
OPEN BURNING
OPEN DUMPING
TRANSFER STATIONS
WASTE PROCESSING
*

INDIRECT HEAT EXCHANGERS

USE HEAT EXCHANGERS

INDIRECT SOURCES

SN A COLLECTIVE TERM FOR BUILDINGS, FACILITIES, AND
INSTALLATIONS, THE EXISTENCE OR USE OF WHICH LEADS
TO AIR POLLUTANT EMISSIONS; E.G., SHOPPING CENTERS,
AMUSEMENT AND RECREATION AREAS, PARKING LOTS, OFFICES.
BT AIR POLLUTION SOURCES
NT AIRPORTS
ROADS
RT *

INDUSTRIAL COOLING

BT AIR POLLUTION SOURCES
RT *

INDUSTRIAL WASTES

BT EFFLUENTS
NT PROCESS WASTE WATER
RT HAZARDOUS MATERIALS
SEWAGE

INITIATING EXPLOSIVES

BT EXPLOSIVES
RT *

INORGANIC COMPOUNDS

NT ACIDS
NITRIC ACID
SULFURIC ACID
AMMONIA
AMMONIA NITROGEN
BORON
CHLORINE

CYANIDES
FLUORIDES
HYDROGEN
 HYDROGEN FLUORIDE
 HYDROGEN SULFIDE
IRON
KAOLINITE
MICA
NITROGEN
 NITRIC ACID
 NITROGEN OXIDES
 NITROGEN DIOXIDE
NONFERROUS METALS
 ARSENIC
 BARIUM
 BERYLLIUM
 CADMIUM
 CHROMIUM
 COPPER
 LEAD
 MANGANESE
 MERCURY
 NICKEL
 SILVER
 SODIUM
 ZINC
PHOSPHORUS
SELENIUM
SILICATES
 ASBESTOS
 FELDSPARS
SULFUR
 ALKYL BENZENE SULFONATES
 SULFUR OXIDES
 SULFUR DIOXIDE
 SULFURIC ACID
RT CHEMICAL MANUFACTURING

INSECTICIDES

USE PESTICIDES

INTERNAL COMBUSTION ENGINES

BT AIR POLLUTION SOURCES
 POWER SOURCES
POINT SOURCES
 POWER SOURCES
NT DIESEL ENGINES
 GASOLINE ENGINES
RT NUCLEAR ENERGY

STEAM GENERATING PLANTS
TURBINES
VEHICLES

IRON

BT INORGANIC COMPOUNDS
POINT SOURCES
RT FERROALLOYS
*

JET THRUST UNITS

BT EXPLOSIVES
RT *

JUNKYARDS

BT WASTE DISPOSAL
RT DUMPING GROUNDS
GARBAGE COLLECTION
INCINERATORS
LANDFILLS
OPEN BURNING
OPEN DUMPING
TRANSFER STATIONS
WASTE PROCESSING

KAOLINITE

BT INORGANIC COMPOUNDS
RT *

KEY LARGO CORAL REEF PRESERVE, FL

KWAJELEIN ATOLL

LAKES

NT LAKES, SPECIFIC
RT WATERWAYS

LAKES, SPECIFIC

SN A COLLECTIVE TERM FOR SPECIFIC LAKES WHICH HAVE
BEEN TREATED INDIVIDUALLY IN THE REGULATIONS AND
DATA BASE; NAMES OF LAKES ARE NOT LISTED IN
THE THESAURUS.
BT LAKES

LAND ACQUISITION

LAND CLASSIFICATION

BT CLASSIFICATION
RT AIR QUALITY CLASSIFICATION

WATER QUALITY CLASSIFICATION

LAND PRESERVATION

RT FOREST PRESERVATION

LANDFILLS

SN SITES FOR DISPOSAL OF SOLID WASTES ON LAND BY
COVERING; SITES OR DISPOSAL PROCEDURES USED ARE
INADEQUATE FOR SANITARY DISPOSAL OF HAZARDOUS
OR PUTRESCIBLE WASTES.
BT AIR POLLUTION SOURCES
WASTE DISPOSAL
NT SANITARY LANDFILL
RT DUMPING GROUNDS
GARBAGE COLLECTION
INCINERATORS
JUNKYARDS
OPEN BURNING
OPEN DUMPING
TRANSFER STATIONS
WASTE PROCESSING
*

LEAD

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

LIQUID FUELS

BT FUELS
NT FUEL OIL
GASOLINE
RT COAL
COKE
OILS
WOOD

LONG ISLAND, NY

LOW EXPLOSIVES

BT EXPLOSIVES
RT *

LUMBER

SN WOOD USED AS A SOURCE OF BUILDING MATERIAL.

BT POINT SOURCES
RT PULP MILLS
WOOD
*

MANGANESE

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

MANUFACTURING

BT AIR POLLUTION SOURCES
NT CHEMICAL MANUFACTURING
RT POINT SOURCES
*

MAXIMUM PERMISSIBLE CONCENTRATION

SN TERM USED ONLY FOR RADIATION STANDARDS.
BT RADIATION STANDARDS.
RT MAXIMUM PERMISSIBLE DOSE

MAXIMUM PERMISSIBLE DOSE

SN TERM USED ONLY FOR RADIATION STANDARDS.
BT RADIATION STANDARDS
RT MAXIMUM PERMISSIBLE CONCENTRATION

MEASUREMENTS

SN TERM FOR MEASUREMENTS OR MEASUREMENT METHODS
REQUIRED FOR A PARTICULAR POLLUTANT, EMISSION,
OR EFFLUENT.

MERCURY

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

METHYLENE BLUE

BT ORGANIC COMPOUNDS
RT *

MICA

BT	INORGANIC COMPOUNDS
RT	*
MIDWAY ISLANDS	
MISTS	
BT	AIRBORNE PARTICULATES
RT	ASH
	DUST
	FUMES
	SMOKE
MIXING ZONE	
SN	AN AREA OF WATER TO WHICH EFFLUENTS, INCLUDING HEAT, MAY BE DISCHARGED FOR DISPERSAL.
RT	EFFLUENTS
MONITORING	
NT	STACK MONITORING
NICKEL	
BT	AIR POLLUTION SOURCES
	NONFERROUS METALS
	INORGANIC COMPOUNDS
	NONFERROUS METALS
	POINT SOURCES
	NONFERROUS METALS
RT	*
NITRATES	
RT	NITRITES
	NITROGEN
NITRIC ACID	
BT	INORGANIC COMPOUNDS
	ACIDS
	NITROGEN
RT	NITROGEN OXIDES
	SULFURIC ACID
NITRITES	
RT	NITRATES
	NITROGEN
NITROGEN	
BT	INORGANIC COMPOUNDS
NT	NITRIC ACID
	NITROGEN OXIDES
	NITROGEN DIOXIDE

RT	NITRATES NITRITES *
NITROGEN DIOXIDE	
BT	INORGANIC COMPOUNDS NITROGEN NITROGEN OXIDES
NITROGEN OXIDES	
BT	INORGANIC COMPOUNDS NITROGEN
NT	NITROGEN DIOXIDE
RT	NITRIC ACID OXIDANTS
NOISE	
NT	NOISE CONTROL NOISE LEVELS
NOISE CONTROL	
BT	NOISE
RT	NOISE LEVELS
NOISE LEVELS	
BT	NOISE
RT	NOISE CONTROL
NONEXPLOSIVE AMMUNITION	
BT	EXPLOSIVES
RT	*
NONFERROUS METALS	
BT	AIR POLLUTION SOURCES INORGANIC COMPOUNDS POINT SOURCES
NT	ARSENIC BARIUM BERYLLIUM CADMIUM CHROMIUM COPPER LEAD MANGANESE MERCURY NICKEL SILVER SODIUM ZINC

RT	HAZARDOUS MATERIALS SMELTERS *
NUCLEAR ENERGY	
BT	AIR POLLUTION SOURCES POWER SOURCES POINT SOURCES POWER SOURCES
RT	INTERNAL COMBUSTION ENGINES STEAM GENERATING PLANTS TURBINES
ODORS	
OIL SPILLS	
BT	ACCIDENTS ORGANIC COMPOUNDS OILS
RT	DISPERSANTS OIL STORAGE OIL TRANSFER SOLVENTS
OIL STORAGE	
BT	ORGANIC COMPOUNDS OILS
RT	STORAGE OIL SPILLS OIL TRANSFER
OIL TRANSFER	
BT	ORGANIC COMPOUNDS OILS
RT	OIL SPILLS OIL STORAGE
OILS	
BT	ORGANIC COMPOUNDS
NT	OIL SPILLS OIL STORAGE OIL TRANSFER
RT	HAZARDOUS MATERIALS LIQUID FUELS PETROLEUM REFINERIES SALVAGE *

OPACITY

RT AIRBORNE PARTICULATES

OPEN BURNING

BT AIR POLLUTION SOURCES
WASTE DISPOSAL
RT DUMPING GROUNDS
FIRES
GARBAGE COLLECTION
INCINERATORS
JUNKYARDS
LANDFILLS
OPEN DUMPING
TRANSFER STATIONS
WASTE PROCESSING
*

OPEN DUMPING

BT WASTE DISPOSAL
RT DUMPING GROUNDS
GARBAGE COLLECTION
INCINERATORS
JUNKYARDS
LANDFILLS
OPEN BURNING
TRANSFER STATIONS
WASTE PROCESSING

ORGANIC CARBON

UF TOC
TOTAL ORGANIC CARBON
BT ORGANIC COMPOUNDS
CARBON
RT CARBON MONOXIDE

ORGANIC COMPOUNDS

NT ALCOHOLS
ALDEHYDES
CARBON
CARBON MONOXIDE
ORGANIC CARBON
CCE
ETHYLENE
HYDROCARBONS
METHYLENE BLUE
OILS
OIL SPILLS
OIL STORAGE
OIL TRANSFER

	RT	PHENOLS CHEMICAL MANUFACTURING
OXIDANTS		
	RT	CARBON MONOXIDE NITROGEN OXIDES PHOTOCHEMICAL REACTIONS SULFUR OXIDES
PACIFIC OCEAN		
	RT	COASTS SALINE WATER WATERWAYS WETLANDS
PACKAGING		
	RT	CONTAINERS EXPLOSIVES RADIOACTIVE SUBSTANCES STORAGE TANKS
PARTICULATES		
		USE AIRBORNE PARTICULATES
PERMITS		
	SN	LICENSES REQUIRED FOR THE CONSTRUCTION OR OPERATION OF A FACILITY OR THE PERFORMANCE OF SOME ACT.
PEST CONTROL		
	BT	PESTS
	RT	HERBICIDES PESTICIDES
PESTICIDES		
	UF	ECONOMIC POISONS INSECTICIDES
	BT	PESTS
	RT	AGRICULTURAL POLLUTION HAZARDOUS MATERIALS HERBICIDES PEST CONTROL
PESTS		
	NT	HERBICIDES PEST CONTROL PESTICIDES
	RT	WILDLIFE
PETROLEUM		

UF	CRUDE OIL
BT	POINT SOURCES
RT	COAL
	COKE
	OILS
	REFINERIES
	SALVAGE
	*
PH	
UF	HYDROGEN ION CONCENTRATION
RT	HYDROGEN
PHENOLS	
BT	ORGANIC COMPOUNDS
RT	*
PHOSPHORUS	
BT	INORGANIC COMPOUNDS
RT	*
PHOTOCHEMICAL REACTIONS	
RT	OXIDANTS
PLANT LIFE	
	USE FLORA
PLASTICS AND SYNTHETICS	
UF	SYNTHETICS
BT	POINT SOURCES
NT	VINYL CHLORIDES
RT	*
POINT SOURCES	
SN	MANUFACTURING POINT SOURCE CATEGORY; PROCESSES AND SUBSTANCES CAUSING WATER POLLUTION, FOR WHICH THE FEDERAL GOVERNMENT HAS ESTABLISHED EFFLUENT STANDARDS.
NT	ASBESTOS
	BOILERS
	CEMENT PLANTS
	CHEMICAL MANUFACTURING
	COATINGS
	COKE OVENS
	FEEDLOTS
	FERROALLOYS
	STEEL
	FERTILIZERS
	FURNACES

BLAST FURNACES
CUPOLAS
GRAIN HANDLING
IRON
LUMBER
NONFERROUS METALS
 ARSENIC
 BARIUM
 BERYLLIUM
 CADMIUM
 CHROMIUM
 COPPER
 LEAD
 MANGANESE
 MERCURY
 NICKEL
 SILVER
 SODIUM
 ZINC
PETROLEUM
PLASTICS AND SYNTHETICS
 VINYL CHLORIDES
POWER SOURCES
 INTERNAL COMBUSTION ENGINES
 DIESEL ENGINES
 GASOLINE ENGINES
 NUCLEAR ENERGY
 STEAM GENERATING PLANTS
 TURBINES
PULP MILLS
REFINERIES
RUBBER
SINTERING
RT EFFLUENTS
MANUFACTURING

POTABLE WATER
UF DRINKING WATER

POWER SOURCES
BT AIR POLLUTION SOURCES
POINT SOURCES
NT INTERNAL COMBUSTION ENGINES
 DIESEL ENGINES
 GASOLINE ENGINES
NUCLEAR ENERGY
STEAM GENERATING PLANTS
TURBINES
RT *

PROCESS WASTE WATER

BT EFFLUENTS
INDUSTRIAL WASTES

PROPELLANT EXPLOSIVES

BT EXPLOSIVES
RT *

PROTECTED SPECIES

NT ENDANGERED SPECIES
THREATENED SPECIES
RT AQUATIC LIFE
FLORA
WILDLIFE

PULP MILLS

BT AIR POLLUTION SOURCES
POINT SOURCES
RT LUMBER
WOOD
*

RADIATION SOURCES

RADIATION STANDARDS

NT MAXIMUM PERMISSIBLE CONCENTRATION
MAXIMUM PERMISSIBLE DOSE

RADIOACTIVE SUBSTANCES

RT HAZARDOUS MATERIALS
PACKAGING
STORAGE
TRANSPORTATION
WASTE DISPOSAL

RECORD KEEPING

SN REQUIRED RECORDING AND FILING OF DATA FOR POSSIBLE
INSPECTION BY A SUPERVISING AGENCY.
RT REPORTING REQUIREMENTS

REFINERIES

BT POINT SOURCES
RT OILS
PETROLEUM
*

REFUSE

UF SOLID WASTE
RT WASTE DISPOSAL

RENDERING

RT AGRICULTURAL POLLUTION

REPORTING REQUIREMENTS

SN REQUIREMENTS THAT REPORTS BE FILED WITH A
SUPERVISORY AGENCY, EITHER AS A PART OF NORMAL
OPERATIONS OR AFTER AN ACCIDENT.

RT RECORD KEEPING

RESERVOIRS

USE IMPOUNDMENTS OF WATER

RIVERS

UF STREAMS
NT RIVERS, SPECIFIC
RT WATERWAYS

RIVERS, SPECIFIC

SN A COLLECTIVE KEYWORD FOR SPECIFIC RIVERS WHICH
HAVE BEEN TREATED INDIVIDUALLY IN THE REGULATIONS
AND DATA BASE; NAMES OF RIVERS ARE NOT LISTED
IN THE THESAURUS.

BT RIVERS

ROADS

BT AIR POLLUTION SOURCES
INDIRECT SOURCES

RT AIRPORTS

ROCKET AMMUNITION

BT EXPLOSIVES
RT *

ROCKET MOTORS

BT EXPLOSIVES
RT *

RUBBER

BT POINT SOURCES
RT *

SALINE WATER

RT ATLANTIC OCEAN
COASTS
PACIFIC OCEAN
TIDAL WATER
WETLANDS

SALTS

SALVAGE

RT OILS
PETROLEUM

SANITARY LANDFILL

SN SITES FOR NONPOLLUTING DISPOSAL OF SOLID WASTES
ON THE LAND, BY SPREADING WASTES IN LAYERS,
COMPACTING THEM TO THE SMALLEST PRACTICAL VOLUME,
AND COVERING THEM WITH SOIL DAILY.
BT AIR POLLUTION SOURCES
LANDFILLS
WASTE DISPOSAL
LANDFILLS

SCUM

SEAPORTS

RT BAYS
COASTS

SEDIMENTATION

RT DEPOSITION
EROSION
SETTLEABLE SOLIDS

SEDIMENTS

USE SETTLEABLE SOLIDS

SELENIUM

BT INORGANIC COMPOUNDS
RT *

SEPARATION PROCESSES

BT AIR POLLUTION SOURCES
RT *

SETTLEABLE SOLIDS

UF SEDIMENTS
RT DEPOSITION
DISSOLVED SOLIDS
EROSION
SEDIMENTATION
SUSPENDED SOLIDS

SEWAGE

BT EFFLUENTS
RT HAZARDOUS MATERIALS
INDUSTRIAL WASTES
SLUDGE

SEWAGE DISPOSAL

NT SEWER SYSTEMS
WATER TREATMENT WORKS

SEWER SYSTEMS

SN NETWORKS OF SEWER PIPES.
BT SEWAGE DISPOSAL
ER WATER TREATMENT WORKS

SHELLFISH

USE FISH

SILICATES

BT INORGANIC COMPOUNDS
NT ASBESTOS
FELDSPARS
RT *

SILVER

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS
RT *

SINTERING

BT AIR POLLUTION SOURCES
POINT SOURCES
RT *

SLUDGE

RT HAZARDOUS MATERIALS
SEWAGE

SMELTERS

BT AIR POLLUTION SOURCES
RT FERROALLOYS
NONFERROUS METALS
*

SMOKE

BT AIRBORNE PARTICULATES
RT ASH
DUST
FUMES
MISTS

SODIUM

BT AIR POLLUTION SOURCES
NONFERROUS METALS
INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS

SOLID WASTE

USE REFUSE

SOLVENTS

RT DISPERSANTS
HAZARDOUS MATERIALS
OIL SPILLS

SPRAYING

BT AIR POLLUTION SOURCES
RT COATINGS
*

STACK MONITORING

SN CONTINUOUS MEASUREMENT OF STACK EMISSIONS.
BT MONITORING
RT STACK TESTS

STACK TESTS

SN OCCASIONAL MEASUREMENTS OF STACK EMISSIONS.
BT TESTS
RT STACK MONITORING

STARTER CARTRIDGES

BT EXPLOSIVES
RT *

STEAM GENERATING PLANTS

BT AIR POLLUTION SOURCES
POWER SOURCES
POINT SOURCES
POWER SOURCES
RT INTERNAL COMBUSTION ENGINES
NUCLEAR ENERGY
TURBINES

STEEL

BT AIR POLLUTION SOURCES
FERROALLOYS
POINT SOURCES
FERROALLOYS

STOCKPILES

SN SUPPLIES OF MATERIALS STORED IN THE OPEN, WHICH
COULD CAUSE FUGITIVE DUST.
BT AIR POLLUTION SOURCES
RT *

STORAGE

NT OIL STORAGE
RT EXPLOSIVES
RADIOACTIVE SUBSTANCES

STORAGE TANKS

RT CONTAINERS
PACKAGING

STREAMS

USE RIVERS

SULFATES

RT SULFUR

SULFITES

RT SULFUR

SULFUR

BT INORGANIC COMPOUNDS
NT ALKYL BENZENE SULFONATES
SULFUR OXIDES
SULFUR DIOXIDE
SULFURIC ACID
RT SULFATES
SULFITES
*
*

SULFUR DIOXIDE

BT INORGANIC COMPOUNDS
SULFUR
SULFUR OXIDES

SULFUR OXIDES

BT INORGANIC COMPOUNDS
SULFUR
NT SULFUR DIOXIDE
RT ALKYL BENZENE SULFONATES
OXIDANTS
SULFURIC ACID

SULFURIC ACID

BT INORGANIC COMPOUNDS
ACIDS
SULFUR
RT ALKYL BENZENE SULFONATES
NITRIC ACID
SULFUR OXIDES

SUSPENDED SOLIDS
RT DISSOLVED SOLIDS
SETTLEABLE SOLIDS

SYNTHETICS
USE PLASTICS AND SYNTHETICS

TASTE

TEMPERATURE
RT THERMAL POLLUTION

TESTS
NT STACK TESTS

THERMAL POLLUTION
RT EFFLUENTS
TEMPERATURE

THREATENED SPECIES
BT PROTECTED SPECIES
RT ENDANGERED SPECIES

TIDAL WATER
SN WATER AFFECTED BY THE TIDES; WATERS ARE OF VARYING
SALINITY.
NT ESTUARIES
RT COASTS
SALINE WATER
WETLANDS

TOC
USE ORGANIC CARBON

TOTAL ORGANIC CARBON
USE ORGANIC CARBON

TOXIC SUBSTANCES
SN TERM USED IF A SPECIFIC TOXIC SUBSTANCE IS NOT
LISTED IN THE ABSTRACT AND/OR THESAURUS.
RT HAZARDOUS MATERIALS

TRANSFER STATIONS

SN SUPPLEMENTAL TRANSPORTATION FACILITIES USED TO
TRANSFER SOLID WASTES FROM SMALL VEHICLES TO
LARGER ONES.
BT WASTE DISPOSAL
RT DUMPING GROUNDS
GARBAGE COLLECTION
INCINERATORS
JUNKYARDS
LANDFILLS
OPEN BURNING
OPEN DUMPING
WASTE PROCESSING

TRANSPORTATION

RT EXPLOSIVES
RADIOACTIVE SUBSTANCES

TURBIDITY

TURBINES

BT AIR POLLUTION SOURCES
POWER SOURCES
POINT SOURCES
POWER SOURCES
RT INTERNAL COMBUSTION ENGINES
NUCLEAR ENERGY
STEAM GENERATING PLANTS

URBAN AREAS

UF CITIES
NT URBAN AREAS, SPECIFIC

URBAN AREAS, SPECIFIC

SN A COLLECTIVE KEYWORD FOR SPECIFIC URBAN AREAS
WHICH HAVE BEEN TREATED INDIVIDUALLY IN THE
REGULATIONS AND DATA BASE; NAMES OF CITIES
ARE NOT LISTED IN THE THESAURUS.
UF CITIES
BT URBAN AREAS

VARIANCE

SN LICENSE TO ENGAGE IN AN ACT CONTRARY TO THE RULE.

VEHICLES

BT AIR POLLUTION SOURCES
RT AIRCRAFT
INTERNAL COMBUSTION ENGINES
WATERCRAFT

*

VINYL CHLORIDES

BT CHLORIDES
POINT SOURCES
PLASTICS AND SYNTHETICS

VOLATILE SUBSTANCES

Rt HAZARDOUS SUBSTANCES

WAKE ISLAND

WASTE DISPOSAL

NT DUMPING GROUNDS
GARBAGE COLLECTION
INCINERATORS
CONICAL BURNERS
JUNKYARDS
LANDFILLS
SANITARY LANDFILL
OPEN BURNING
OPEN DUMPING
TRANSFER STATIONS
WASTE PROCESSING
RT RADIOACTIVE SUBSTANCES
REFUSE

WASTE PROCESSING

SN REFUSE TREATMENT METHODS, INCLUDING SHREDDING,
BALING, RECYCLING, AND COMPOSTING.
BT WASTE DISPOSAL
RT DUMPING GROUNDS
GARBAGE COLLECTION
INCINERATORS
JUNKYARDS
LANDFILLS
OPEN BURNING
OPEN DUMPING
TRANSFER STATIONS

WATER POLLUTION CONTROL

SN DEVICE OR PROCEDURE USED TO LIMIT THE RELEASE OF
EFFLUENTS INTO THE WATER.

WATER QUALITY CLASSIFICATION

BT CLASSIFICATION
RT AIR QUALITY CLASSIFICATION
LAND CLASSIFICATION

WATER QUALITY STANDARDS

RT EFFLUENT STANDARDS

WATER RIGHTS

SN THE RIGHT TO DRAW WATER FROM A SOURCE, INCLUDING
GROUNDWATER SOURCES.

WATER TREATMENT WORKS

SN SEWAGE TREATMENT FACILITIES.

BT SEWAGE DISPOSAL

RT SEWER SYSTEMS

WATERCRAFT

RT AIRCRAFT
VEHICLES

WATERWAYS

SN BODIES OF WATER USED FOR WATERCRAFT NAVIGATION.

RT ATLANTIC OCEAN

CHANNELS

LAKES

PACIFIC OCEAN

RIVERS

WETLANDS

RT ATLANTIC OCEAN
COASTS
PACIFIC OCEAN
SALINE WATER
TIDAL WATER

WILDLIFE

RT AQUATIC LIFE
FLORA
PESTS
PROTECTED SPECIES

WOOD

BT FUELS
RT COAL
COKE
FOREST PRESERVATION
LIQUID FUELS
LUMBER
PULP MILLS

ZINC

BT AIR POLLUTION SOURCES
NONFERROUS METALS

INORGANIC COMPOUNDS
NONFERROUS METALS
POINT SOURCES
NONFERROUS METALS

RT *

* CHECK THE BROADER TERMS FOR A LIST OF POTENTIALLY RELATED TERMS

Appendix C - Source Code and Subroutines

/*

REPAIR

Repair updates the laws.toc file for CELDS, creates and/or adds to the isol files, and performs minor error checking functions on the laws files.

Arguments: - if isol files are to be created
 <fname> for laws files to be repaired
 (any number of laws files may be named)

Main variables:

acc:	current accession number
agy:	file descriptor of agy.isol file
argc:	number of arguments with procedure was called
argnum:	argument number of file undergoing repair
argstart:	argument number of first file to undergo repair
argv	array of pointers to arguments
att	file descriptor of att.isol file
bf	buffer for reading from laws file
card	copy of line read from laws file
ch_this_fil:	running total of characters read from laws file
chars:	number of chars read by current read
data:	file descriptor of file undergoing repair
gps:	file descriptor of gps.isol file
installing:	a flag set to 1 if toc file is to be modified, otherwise set to 0.
iobuf:	buffer used to read records from the laws files
isols:	a flag set to 1 if isol files are to be produced, otherwise set to 0.
top:	file descriptor of top.isol file
mec:	file descriptor of mec.isol file
p:	pointer into read buffer (bf)
prev_acc:	previous accession number
prev_stop:	previous stop code
prev_type:	previous field number
q:	pointer into copy of line read (card)
stop:	current stop code
toc:	file descriptor of laws.toc file
type:	current field number
type_start:	record to be written into laws.toc file

*/

```
main(argc,argv) int argc; char *argv[];
```

```
{
```

```
/* Declaration of variables */
```

```
char bf[82], card[82], *ch_this_fil, iobuf[530];  
int acc, agy, argnum, argstart, att, chars, data, gps,  
    installing, isols, top, mec, prev_acc, prev_stop,  
    prev_type, stop, toc, type, type_start[13];  
register char *p, *q;
```

```

/* Check for proper calling of repair */
    if (argc < 2) { printf("USAGE: repair <lawsfile>\n"); return; }

toc = open("laws.toc",1);
/* If toc file doesn't exist, then turn installing off. This
   means that repair is being used only for error-checking. */
    if (toc < 0) installing = 0;
    else installing = 1;

/* Set isols flag */
    isols = (*argv[1] == '-' ? 1:0);

if (isols)
{
    mec = open("mec.isol",1);
    if (mec < 0)
        /* create all of the isol files */
        {
            mec = creat("mec.isol",0666);
            if (mec < 0) {perror("creating isol files"); return;}
            gps = creat("gps.isol",0666);
            agy = creat("agy.isol",0666);
            att = creat("att.isol",0666);
            top = creat("top.isol",0666);
        }
    else
        /* Files already exist, append to end */
        {
            seek(mec,0,2);
            gps = open("gps.isol",1);
            seek(gps,0,2);
            agy = open("agy.isol",1);
            seek(agy,0,2);
            att = open("att.isol",1);
            seek(att,0,2);
            top = open("top.isol",1);
            seek(top,0,2);
        }
}

/* Set argstart to argument number of first file */
    argstart = (isols ? 2:1);

/* Now, loop for each file to be repaired (up to argc) */
for (argnum = argstart; argnum < argc; argnum++)
{
    /* Open laws file to be repaired */
    data = gopen(argv[argnum],&iobuf);
    if (data < 0) {perror("repair"); return;}
    printf("repairing %s\n",argv[argnum]);
}

```

```

/* Initialize parameters */
p = argv[argnum] + 5;
type_start[0] = atoi(p);
prev_stop = 2;
prev_type = 12;
prev_acc = type_start[0];
ch_this_fil = 0;

/* Now, read and check data until entire file has been read */
do
{
/* Read 1 line from file into bf */
chars = ggets(bf,&iobuf);

/* Set pointers, p and q */
p = &bf[0];
q = &card[0];

/* Copy bf into card */
while (*q++ = *p++);

/* Convert stop code, field number, and acc number to integer */
bf[10] = 0;
stop = atoi(&bf[8]);
bf[8] = 0;
type = atoi(&bf[5]);
bf[5] = 0;
acc = atoi(&bf[0]);

if (acc != prev_acc || chars <= 0)
/* Then this is either a new law or an error */
{
if (acc < prev_acc && chars > 0) printf("bad acc :%s\n",card);
if (installing)
{
lseek(toc, (prev_acc * 26.0));
write(toc,type_start,26);
}
}

/* Check for end of file */
if (chars <= 0) break;

if (type != prev_type)
/* Then this is a new field (and maybe a new law) */
{
if (type != (prev_type + 1) && type != 1)
printf("bad type:%s\n",card);
if (type == 1 && (prev_stop != 2 || prev_type != 12))
printf("bad fld1:%s\n",card);
if (stop == 2 && type != 12)

```

```

        printf("bad stop:%s\n",card);
    if (prev_stop < 1) printf("bad stop:%s\n",card);
    if (type >= 1 && type <= 12) type_start[type] = ch_this_fil;
        else printf("bad type:%s\n",card);
    }

    if (isols)
    /* Then add to isol files */
    {
        /* Replace nul at end of card with a newline */
        card[chars - 1] = 012;

        switch(type)
        {
            case 5: write(mec,card,chars); break;
            case 6: write(gps,card,chars); break;
            case 7: if (type != prev_type)
                    write(agy,card,chars); break;..
            case 11: write(att,card,chars); break;
            case 12: write(top,card,chars); break;
            default: break;
        }
    }

    /* Change this card to previous card */
    prev_stop = stop;
    prev_type = type;
    prev_acc = acc;

    /* Add characters read to running total */
    ch_this_fil += chars;

    } while (chars > 0);

    /* Close laws file */
    close(data);
}

/* Close all open files */
if (installing) close(toc);
if (isols)
{
    close(mec);
    close(gps);
    close(agy);
    close(att);
    close(top);
}
}

```

Appendix D - The Make_Search Subroutine

```

#
/*
                                MAKE_SEARCH

    Make_search takes an "alpha" file for a given field and creates
    the search file for that field.

Arguments:      mec, gps, agy, key, or att

Main variables:
    alpha:      file descriptor for alpha file
    bf:         buffer used for reading from alpha file
    collisions: total number of hash collisions
    eof:        end-of-file indicator
    first_hash: actual hash number for value
    hash_num:   hash number incremented to avoid collision
    hash_table: array, 4011 long
    i:          counter
    iobuf:      structure used for reading from alpha file.
                it is of the shape required by ggets.
                the ggets routines are the only ones that
                touch these variables.
    laws:       count of laws with the current value
    list_file:  file descriptor for list file
    loc_in_laws_list: start of current law list in list file
    old_value:  copy of last value read
    p:         pointer into bf
    q:         pointer into old_value
    v:         structure of shape val_record
    val_file:  file descriptor for val file
    value_num:

*/

#include "search.i"

main(argc,argv) int argc; char *argv[];
{
    /* Declaration of variables */
    char bf[122], old_value[64], *p, *q;
    int alpha, collisions, eof, first_hash, hash_num,
        hash_table[hash_size], i, law[2000], laws, list_file,
        loc_in_laws_list, val_file, value_num;
    struct iostru {
        int gfildes;
        char *gnextp;
        char *gstop;
        int geof;
        char gdbuf[513];
    } iobuf;
    struct val_record v;

    /* Check for proper calling of make_search */

```

```

        if (argc != 2)
        {
            printf("USAGE: make_search <field_name>\n");
            return;
        }

    printf("making search file for field %s\n",argv[1]);

    concat(argv[1],".alpha",bf);
    /* Open appropriate alpha file */
    alpha = fopen(bf,&iobuf);
    if (alpha < 0) {perror("mksrch(oal)"); return;}

    concat(argv[1],".val",bf);
    /* Create appropriate val file */
    val_file = creat(bf,0666);
    if (val_file < 0) {perror("mksrch(cva)"); return;}

    concat(argv[1],".list",bf);
    /* Open appropriate list file */
    list_file = creat(bf,0666);
    if (list_file < 0) {perror("mksrch(cli)"); return;}

    /* Zero out hash table and save room in val file */
    for (i = 0; i < hash_size; i++) hash_table[i] = 0;
    write(val_file,hash_table, hash_size * 2);

    /* Initialize counters */
    value_num = 0;
    collisions = 0;
    loc_in_laws_list = 0;

    /* Read first line from alpha into bf */
    eof = ggets(bf,&iobuf);
    if (eof <= 0) {perror("mksrch(ral)"); return;}

    /* Now, repeat until end-of-file is reached */
    while (eof > 0)
    {
        /* copy the value into old_value */
        p = &bf[10];
        q = old_value;
        while(*q++ = *p++);

        /* Zero out the unused part of val_record */
        while (q < &old_value[63]) *q++ = 0;

        /* Initialize laws counter */
        laws = 0;

        /* Repeat as long as value remains the same */

```

```

while (eof > 0 && compar(&bf[10],old_value) == 0)
{
    /* Convert accession number to integer and put in laws */
    bf[5] = 0;
    law[laws++] = atoi(&bf[0]);

    /* Read the next record */
    eof = ggets(bf,&iobuf);
}

/* Mark end of law list with 19999 */
law[laws++] = 19999;

/* Write list of laws into list file */
write(list_file,law, laws << 1);

/* Compute hash number */
hash_num = hash(old_value) % max_hash_num;
first_hash = hash_num;

/* Add 1 as long as this hash_num already exists */
while(hash_table[hash_num] != 0)
{
    collisions++;
    hash_num++;
}

/* Print message for collisions */
if (first_hash != hash_num)
    printf("slide %d to %d\n",first_hash,hash_num);

/* Put value_num into slot in the hash table */
hash_table[hash_num] = ++value_num;

/* Print insertion message */
printf("%5d at %5d:%s\n",laws,hash_num,old_value);

/* Copy this value into val_record array */
p = old_value;
q = v.value;
while (p < &old_value[63]) *q++ = *p++;

/* Set first_law to beginning of law list */
v.first_law = loc_in_laws_list;

/* Write val_record into val file */
write(val_file,&v,64);

/* Adjust loc_in_laws_list */
loc_in_laws_list += laws;

```

AD-A061 158

CONSTRUCTION ENGINEERING RESEARCH LAB (ARMY) CHAMPAI--ETC F/G 5/2
SYSTEM DOCUMENTATION FOR COMPUTER-AIDED ENVIRONMENTAL LEGISLATION--ETC(U)
SEP 78 R L WELSH
CERL-SR-N-31

UNCLASSIFIED

2 OF 2
AD
A061158

NL



END
DATE
FILMED

1-79
DDC

```

    }

    /* Go to beginning of file and write hash table */
    seek(val_file, 0, 0);
    write(val_file, hash_table, hash_size * 2);

    /* Close all open files */
    close(alpha);
    close(list_file);
    close(val_file);

    /* Print out total statistics */
    printf(" total collisions = %d\n", collisions);
    printf(" unique values = %d\n", value_num);
    printf(" last list word = %d\n", loc_in_laws_list);
}

```

Appendix E - The Hier Subroutine

```

#
/*
                                H I E R

        Hier put in the hierarchy of terms for CELDS.  It reads the file
        "key.hier" and constructs the hierarchical terms.
*/
#include "search.i"
/* Global variables */
        int loc_in_law_list, val_num;
        int hash_table[hash_size];
        int data, list, nlist, nval, val;
        char iobuf[600];
        struct val_record v;

main()
{
/* Declaration of variables */
        int i, first, flag, new[max_laws], old[max_laws];
        int *o, *n;
        char *p, *r, rvalue[63];
        char *q;

        debugging = 0;
/* Open files */
        data = gopen("/cerl/celds/key.hier",iobuf);
        if(data<0){perror("open(key.hier)");return;}
        list = open("/cerl/celds/top.list",0);
        if(list<0){perror("open(top.list)");return;};
        val = open("/cerl/celds/top.val",0);
        if(val<=0){perror("open(top.val)");return;}
        if (debugging) printf("Files are open\n");

/* Create new files */
        nval = creat("/cerl/celds/key.val",0644);
        if(nval <= 0){perror("creat(key.val)");return;}
        nlist = creat("/cerl/celds/key.list",0644);
        if(nlist <= 0){perror("creat(key.list)");return;}
        if(debugging)printf("Files are created\n");

/* Zero out hash table and write it */
        for (i=0; i<hash_size; i++) hash_table[i] = 0;
        write(nval,hash_table,hash_size*2);
        val_num = 0;
        loc_in_law_list = 0;
        old[0] = 0;

/* Now we're all set to begin */

        first = 1;
        while(ggets(rvalue,&iobuf))
        {

```

```

    r = rvalue;
    if (*r == '^')
    {
        terminate(old);
        /* Zero out v.value */
        for (i=0; i<62; i++) v.value[i] = 0;

        /* Fill "old" array */
        fill(old,&rvalue[1]);

        /* Fill v.value with this term */
        p = &v.value;
        r = &rvalue[1];
        while(*p++ = *r++);
        first = 0;
    }
    else
    {
        flag = fill(new,&rvalue[0]);
        if (flag > 0) or(old,new);
    }
}
terminate(&old);
/* Put final write here */
close(val); close(list); close(nval); close(nlist);
nval = open("/cerl/celds/key.val",0644);
write(nval,hash_table,hash_size*2);
close(nval);
}
/**/
terminate(old) int old[];
{
    /* Declaration of variables */
    int hash_num, i, *p;

    /* Check for nul list */
    if(old[0] == 0 || old[0] == 19999) return;

    /* Find slot in val_file */
    hash_num = hash(v.value) % max_hash_num;
    if(debugging)printf(" %s hashes to %d\n",v.value,hash_num);
    while (hash_table[hash_num] != 0) hash_num++;

    /* Put val_location into hash table */
    hash_table[hash_num] = ++val_num;

    /* Write val_record onto file */
    v.first_law = loc_in_law_list;
    write(nval,&v,64);

    /* Now do the list file */

```

```

        p = old;
        i = 0;
        while (*p++ < 19999) i++;
        i++;
        write(nlist,old,i*2);
        loc_in_law_list += i;
        printf("%5d inserted at %d for %s\n",i-1,hash_num,v.value);
        return;
    }
    /**/
    int fill(laws,rvalue)    int laws[]; char *rvalue;
    {
        int number;
        /* Declaration of variables */
        int bf[100], hash_num, *r, seek_location, wrong;
        struct val_record x;

        /* Calculate hash number */
        hash_num = hash(rvalue) % max_hash_num;
        /* Seek to proper slot in hash table -- each hash_num takes 2 bytes */
        seek_location = hash_num * 2;
        seek(val,seek_location,0);

        /* Read from val file */
        read(val,bf,200);

        /* Point r to beginning of record read */
        r = &bf[0];

        /* Initialize wrong to be true */
        wrong = 1;

        while (*r)
        /* If *r is zero, then this hash number contains a zero in the table */
        {
            /* Locate record with this hash number */
            seek_location = ((*r * 64) + val_table_start);
            seek(val,seek_location,0);
            read(val,&x,64);

            /* Compare fvalue with the value in the val file */
            wrong = compar(x.value,rvalue);
            if(debugging)printf("compar %s to %s\n",x.value,rvalue);

            /* If wrong is zero, we found fvalue */
            if (wrong == 0) break;

            r++;
        }
        if (wrong != 0)
        {

```

```

        printf("WARNING:  %s -- no laws found\n",rvalue);
        return(-1);
    }
    /* Calculate location */
    seek_location = x.first_law * 2;
    seek (list,seek_location,0);
    read(list,laws,max_laws);
    return;
}
/**/
int or(old,new) int old[], new[];
{
    int result[2*max_laws];
    int chars;
    register int *r, *s, *t;

    r = old;
    s = new;
    t = &result[0] - 1;
    do
    {
        if (*r < *s) *++t = *r++; else
            {if (*r == *s) r++;
             *++t = *s++;}
    } while (*t < 19999);

    chars = (t - &result[0]) << 1;
    if (chars > lawset_size)
    {
        printf("lawset size exceeds %d laws\n",(max_laws -1));
        printf("only the first %d laws will be used\n",(max_laws -1));
        t = &result[max_laws];
        *t = 19999;
    }
    t = &result[0];
    r = old;
    while (*t < 19999) *r++ = *t++;
    *r = *t;
    return;
}

```

Appendix F - The Push File

```

:                                     P U S H
:
: Enter start date and time
date
: Remove all old isol files
rm *.isol
: Repair all files -- being careful not to have arg list too long
/cerl/programs/celds/push_progs/obj/repair - laws.? laws.?? laws.[123456789]??
/cerl/programs/celds/push_progs/obj/repair - laws.1???
/cerl/programs/celds/push_progs/obj/repair - laws.2[01234]??
/cerl/programs/celds/push_progs/obj/repair - laws.2[56789]??
: Sort isol files into alpha files and execute make_search
sort +0.10 +0.0 -0.5 mec.isol -o mec.alpha
/cerl/programs/celds/push_progs/obj/make_search mec
sort +0.10 +0.0 -0.5 gps.isol -o gps.alpha

/cerl/programs/celds/push_progs/obj/make_search gps
sort +0.10 +0.0 -0.5 agy.isol -o agy.alpha

/cerl/programs/celds/push_progs/obj/make_search agy
sort +0.10 +0.0 -0.5 att.isol -o att.alpha

/cerl/programs/celds/push_progs/obj/make_search att
sort +0.10 +0.0 -0.5 top.isol -o top.alpha

/cerl/programs/celds/push_progs/obj/make_search top
: Get rid of all alpha files
rm *.alpha
/cerl/programs/celds/push_progs/hier
: Include finish date and time
date

```

Appendix G - The Retriever Subroutines

```
#
/*
```

C E L D S

Celds is the retrieval program for the celds system.

Arguments: - if celds is executed with a "-" argument,
 then on termination, "etis -" is executed.
 This is the case when celds is entered
 from etis.

Global variables:

client:	user id for the person using celds
debugging:	flag set to 1 for debugging program. this causes parameters to be printed.
field_is_searchable:	array indicating which of fields 1 to 13 are searchable
fld_name:	alphabetic identifier for field
old_req_location:	pointer to previous position in request array
punctuation:	array indicating which ASCII are recognized punctuation
reading_commands:	1 until user asks to leave celds, then 0
req_location:	pointer to current position in request
request:	the array containing the current line of user input
requests:	total number of celds commands
temp_file_name:	array of scratch file names
word:	the array containing the current word, null term

Main variables:

i:	counter
p:	pointer into punctuation array

```
*/
```

```
#include "search.i" ..
```

```
main(argc,argv) int argc; char *argv[];
```

```
{
```

```
/* Declaration of variables */
```

```
char *p;  
int i;
```

```
/* Set a trap for break */
```

```
signal(2,1);
```

```
/* Print hail message */
```

```
write(2,"\nWelcome to CELDS\n\n",20);  
message();
```

```
/* Zero out punctuation array */
```

```
p = &punctuation[0];  
for (i = 0; i < 128; i++) *p++ = 0;
```

```

/* Fill in only recognized punctuation */
punctuation[0] = 1;
punctuation['&' & 0177] = 1;
punctuation['|' & 0177] = 1;
punctuation[';' & 0177] = 1;
punctuation['"' & 0177] = 1;
punctuation['(' & 0177] = 1;
punctuation[')' & 0177] = 1;
punctuation[' ' & 0177] = 1;

/* Fill in field name abbreviations */
fld_name[1] = "acc";
fld_name[2] = "ttl";
fld_name[3] = "dat";
fld_name[4] = "ref";
fld_name[5] = "mec";
fld_name[6] = "gps";
fld_name[7] = "agy";
fld_name[8] = "bib";
fld_name[9] = "abs";
fld_name[10] = "tbl";
fld_name[11] = "att";
fld_name[12] = "top";
fld_name[13] = "key";

/* Set flags for searchable fields */
field_is_searchable[0] = 0;
field_is_searchable[1] = 1;
field_is_searchable[2] = 0;
field_is_searchable[3] = 0;
field_is_searchable[4] = 0;
field_is_searchable[5] = 1;
field_is_searchable[6] = 1;
field_is_searchable[7] = 1;
field_is_searchable[8] = 0;
field_is_searchable[9] = 0;
field_is_searchable[10] = 0;
field_is_searchable[11] = 1;
field_is_searchable[12] = 1;
field_is_searchable[13] = 1;

/* Initialize everything */
old_req_location = request;
req_location = request;
request[0] = 0;
word[0] = 0;
requests = 0;

/* Turn debugging off */
debugging = 0;

```

```

/* Get user id and put it into client */
    client = getuid() & 0377;

/* Execute this loop until user ends celds session */
do
{
    /* Initialize temp_file name to "celds_tempa" */
    concat("celds_temp","a",temp_file_name);

    /* Set trap for quit */
    signal(2,1);

    /* Set reading_commands to true */
    reading_commands = 1;

    /* Get the next command */
    command();

    /* Increment number of requests */
    requests++;
} while (reading_commands);

/* Remove all of the temp files */
concat("celds_temp","a",temp_file_name);
while (temp_file_name[10] < 'k')
{
    unlink(temp_file_name);
    temp_file_name[10] += 1;
}
unlink("current_laws");
unlink("previous_laws");

/* Print out summary statistics */
/* Print farewell */
printf("Good bye from CELDS\n");

/* Check whether to execute etis */
if (compar(argv[1],"-") == 0) execl("/cerl/etis","etis","-",0);
}

```

```

#
#include "search.i"
/*
                                A B O R T E R

        Aborter is the interrupt procedure that gets called when the
        rubout, del, or break key is pressed. This procedure is only active
        during list, print, and show verbs. At all other times, those keys are
        ignored.

Global variables:      (see list for expr.c)
        reading_commands:      aborter modifies "reading_commands"
*/

int *aborter(){
    reading_commands = 2;
    signal(2,aborter);
    return;
}

```

```
#
#include "search.i"
/*
```

A N D

And takes two lists of law numbers and logically "ands" the two into a third list, i.e. it saves law numbers that occur in both lists.

Arguments	first	file descriptor of file containing first list of law numbers
	second	file descriptor of file containing second list of law numbers
Returns	fil	file descriptor of file containing the "anded" list

And variables:

bfl:	buffer for first list
bf2:	buffer for second list
chars:	number characters written into result
chars1:	number characters read into bfl
chars2:	number characters read into bf2
fil:	file descriptor of resultant list
r:	pointer to result
result:	buffer for resultant list
s:	pointer to bfl
t:	pointer to bf2

```
*/
```

```
int and(first,second) int first,second;
{
/* Declaration of variables */
    int bfl[max_laws], bf2[max_laws], chars, chars1, chars2, fil,
        result[max_laws];
    register int *r,*s,*t;

/* Fill bfl and bf2 with law sets */
    chars1 = read(first,bfl,lawset_size);
    chars2 = read(second,bf2,lawset_size);

/* Guarantee that there is a 19999 at end (paranoia) */
    bfl[chars1 / 2] = 19999;
    bf2[chars2 / 2] = 19999;

/* Set up pointers */
    r = bfl;
    s = bf2;
    t = &result[0];

/* Since t gets pre-incremented the first one will be skipped */
    *t = 0;
```

```

do      {
        if (*r < *s) r++;
        else
            {
                if (*r == *s) *++t = *r++;
                s++;
            }
        } while (*t < 19999);

/* Calculate number of characters in result */
chars = (t - &result[0]) << 1;

/* Create file to write result into */
fil = creat(temp_file_name,0666);
if (fil < 0) {perror("bool(c)"); return(fil);}

/* Write result beginning with result[1] since [0] is skipped */
write(fil,&result[1],chars);

/* Position fil at beginning of file (by closing, then opening) */
close(fil);
fil = open(temp_file_name,0);

/* Increment temp_file_name[10] since we just made a file */
temp_file_name[10] += 1;

/* Close first and second files */
close(first);
close(second);

return(fil);
}

```

```

#
#include "search.i"
/*
                                C O M M A N D

        Command parses the command line to find the verb. Any word that
        is not a verb is taken to be a request to create a new file
        by that name.

Arguments          none

Command variables:
eof:               end of file indicator
laws:             number of laws in list
new:              file descriptor of new file
old:              file descriptor of old file
time_vector:      random number used for insuring uniqueness
                  of job numbers for batch print requests
*/

command()
{
/* Declaration of variables */
    char set_name[62];
    int eof, laws, new, old, time_vector[2];

    if (word[0] == 0)
/* Then no words are waiting */
    {
        eof = get_next_word("What next?:");
        if (eof == 0)
/* Then cntrl-d was typed */
        {
            reading_commands = 0;
            return;
        }
    }

/* Echo command line if debugging is turned on */
    if (debugging) printf("Command: '%s'\n",word);

/* Now, begin compares to find out what command this is */
    if (compar(word,"find") == 0 || compar(word,"for") == 0 ||
        compar(word,"get") == 0)
    {
/* Then this is a new search command */
        get_next_word("Search criterion?:");
        old = expression();
        if (old < 0) return;
        laws = copy_list(old,"current_laws",1);
        close(old);
        printf("%5d laws found\n",laws);
    }
}

```

```

        return;
    }
    /* ---- "and" command ---- */
    if (compar(word,"and") == 0)
    {
        new = open("current_laws",0);
        if (new < 0)
        {
            reject_this_word("no laws selected");
            return;
        }
        get_next_word("And what?:");
        old = expression();
        if (old < 0) {close(new); return;}
        old = and(old,new);
        laws = copy_list(old,"current_laws",1);
        close(old);
        printf("%5d laws remain\n",laws);
        return;
    }

    /* ---- "or" command ---- */
    if (compar(word,"or") == 0)
    {
        new = open("current_laws",0);
        if (new < 0) {reject_this_word("no laws selected"); return;}
        get_next_word("Or what?:");
        old = expression();
        if (old < 0) {close(new); return;}
        old = or(old,new);
        laws = copy_list(old,"current_laws",1);
        close(old);
        printf("%5d laws now selected\n",laws);
        return;
    }

    /* ---- "except" command ---- */
    if (compar(word,"except") == 0)
    {
        new = open("current_laws",0);
        if (new < 0) {reject_this_word("no laws selected"); return;}
        get_next_word("Except what?:");
        old = expression();
        if (old < 0) {close(new); return;}
        old = except(new,old);
        laws = copy_list(old,"current_laws",1);
        close(old);
        printf("%5d laws remain\n",laws);
        return;
    }

```

```

/* ---- "show" command ---- */
if (compar(word,"show") == 0) {show(); return;}

/* ---- "suggest" command ---- */
if (compar(word,"suggest") == 0)
{
    printf("\nType comment (end with cntrl-d):\n\n");
    execute("mail welsh");
    printf("\n");
    get_next_word("");
    return;
}

/* ---- "remove" command ---- */
if (compar(word,"remove") == 0 || compar(word,"delete") == 0)
{
    get_next_word("Set names to delete?:");
    while(word[0])
    {
        /* open it only to see if it exists */
        concat("_",word,set_name);
        old = open(set_name,0);
        if (old < 0)
        {
            printf("%s: no such set, unable to delete\n",word);
        }
        else
        {
            close(old);
            unlink(set_name);
            printf("%s: deleted\n",word);
        }
        get_next_word("");
        if (compar(word,",") == 0) get_next_word("More set names?:");
        if (compar(word,"and") == 0) get_next_word("More set names?:");
    }
    return;
}

/* ---- "save" command ---- */
if (compar(word,"save") == 0)
{
    old = open("current_laws",0);
    if (old < 0)
    {
        reject_this_word("no laws selected");
        return;
    }
    get_next_word("New lawset name?:");
    if (compar(word,"abort") == 0)
    {

```

```

        reject_this_word("lawset not saved");
        return;
    }
    concat("_",word,set_name);
    laws = copy_list(old,set_name,0);
    close(old);
    printf("%5d laws saved\n",laws);
    get_next_word("");
    return;
}

/* ---- "make" command ---- */
if (compar(word,"make") == 0 || compar(word,"set") == 0)
{
    get_next_word("New lawset name?:");
    if (compar(word,"abort") == 0)
    {
        reject_this_word("well if you insist");
        return;
    }
    concat("_",word,set_name);
    get_next_word("Search criterion?:");
    if (compar(word,"is") == 0) get_next_word("Search criterion?:");
    if (compar(word,"from") == 0) get_next_word("Search criterion?:");
    old = expression();
    if (old < 0) return;
    laws = copy_list(old,set_name,0);
    close(old);
    printf("%5d laws saved\n",laws);
    old = open(set_name,0);
    copy_list(old,"current_laws",1);
    close(old);
    return;
}

/* ---- "list" command ---- */
if (compar(word,"list") == 0)
{
    lister(1,"");
    return;
}

/* ---- "print" command ---- */
if (compar(word,"print") == 0)
{
    /* Create the listing in the line printer daemon's directory,
       and give it a pseudo-random name (to avoid conflicts) */

    time(time_vector);
    concat("/usr/lpd/celds",locv(0,time_vector[1]),set_name);
    old = creat(set_name,0666);

```

```

        if (old < 0) {perror("print"); return;}
        laws = lister(old,set_name);
        close(old);
        return;
    }

/* ---- "debug" command ---- */
if (compar(word,"debug") == 0)
{
    debugging = 1 - debugging;
    get_next_word("");
    return;
}

/* ---- "what" command ---- */
if (compar(word,"what") == 0)
{
    get_next_word("Sets, does or is?:");
    if (compar(word,"are") == 0) get_next_word("Say 'the sets':");
    if (compar(word,"the") == 0) get_next_word("Say 'sets':");
    if (compar(word,"sets") != 0)
    {
        reject_this_word("Only 'sets' is available, sorry");
        return;
    }
    execute("ls _*");
    get_next_word("");
    return;
}

/* ---- "help" command ---- */
if (compar(word,"help") == 0)
{
    printf("verbs are:\n");
    printf("FIND,AND,OR,EXCEPT,SAVE,MAKE,OOPS,\n");
    printf("DELETE,SHOW,LIST,PRINT,WHAT,SUGGEST,HELP,END\n");
    get_next_word("More detail?:");
    if (word[0] != 'y') { get_next_word(""); return; }
    printf("FIND    begins a new search\n");
    printf("AND      further limits the previous search\n");
    printf("OR       extends a search\n");
    printf("EXCEPT  excludes selected laws\n");
    printf("SAVE     stores the result of a search\n");
    printf("MAKE     finds and saves\n");
    printf("OOPS     reinstates previous lawset\n");
    printf("\n");
    printf("DELETE   removes a saved lawset\n");
    printf("SHOW     shows the accession numbers of laws found\n");
    printf("LIST     summarizes laws on the terminal\n");
    printf("PRINT    summarizes laws on high-speed printer\n");
    printf("WHAT     shows lawset names\n");
}

```

```

printf("SUGGEST to send a comment to the authors of CELDS\n");
printf("HELP      shows this list\n");
printf("END       signs the user off the system\n");
get_next_word("");
return;
}

/* ---- "end" command ---- */
if (compar(word,"end") == 0 || compar(word,"bye") == 0)
{
    reading_commands = 0;
    return;
}

/* ---- "abort" command ---- */
if (compar(word,"abort") == 0)
{
    reject_this_word("nothing to abort. all is cool.");
    return;
}

/* ---- "oops" command ---- */
if (compar(word,"oops") == 0)
{
    old = open("previous_laws",0);
    if (old < 0) {reject_this_word("Recovery not possible"); return;}
    laws = copy_list(old,"current_laws",0);
    printf("%5d laws recovered\n",laws);
    close(old);
    get_next_word("");
    return;
}

reject_this_word("Oh worthy master, I fear I have\nfailed to
}      understand your intention

```

```

#
#include "search.i"
/*                                C O P Y _ L I S T

    Copy_list copies a list of laws from one file into another.
    All new lawsets are created with a temporary name and are not copied
    until the whole command has been read and checked for syntactic
    correctness. Then copy_list is called to transfer the law list
    into a permanent file.

Argument:      mode      if mode is 1, a copy of what is in the "new"
                        file (if existent) is put into "previous_laws"
                        before old is copied to new.
                        new  name of new file (copied to)
                        old  file descriptor of old file (copied from)

Returns:      -1          for error conditions
                the number of laws copied, for successful calls

Copy_list variables:
    bf:        buffer for reading files
    chars:     number of characters read from a file
    nw:        file descriptor of the new file
    r:         pointer into bf
    tmp:       file descriptor of a temporary file
*/

int copy_list(old,new,mode) int old,mode; char *new;
{
    /* Declaration of variables */
    int bf[max_laws], chars, nw, *r, tmp;

    if (mode == 1)
    /* Then copy "new" to "previous_laws" */
    {
        tmp = open(new,0);
        if (tmp >= 0)
        {
            /* Create "previous_laws" */
            nw = creat("previous_laws", 0666);
            if (nw < 0) {perror("prev_laws"); return(-1);}

            /* Read from new and write to previous */
            chars = read(tmp,bf,lawset_size);
            if (chars < 0 ) {perror("copy"); return(-1);}
            write(nw,bf,chars);
            close(tmp);
            close(nw);
        }
    }
    /* Create new file */

```

```

        nw = creat(new,0666);
        if (nw < 0) {perror("copy"); return(-1);}

/* Read old file */
        chars = read(old,bf,lawset_size);
        if (chars < 0) {perror("copy"); return(-1);}

/* Count number of laws */.....
        bf[chars / 2] = 19999;
        r = bf;
        while (*r++ < 19999);

        chars = (r - &bf[0]) * 2;
        write(nw,bf,chars);
        close(nw);
        return(r - &bf[1]);
    }

```

```

#
#include "search.i"
/*
                                E X C E P T

    Except takes two lists of law numbers and constructs a
    list containing laws that are in the first list and not in the
    second list.

Arguments      first      file descriptor of file containing
                        first list of law numbers
                second     file descriptor of file containing
                        second list of law numbers

Returns        fil        file descriptor of file containing
                        the "ored" list

Or variables:
    bfl:        buffer for first list
    bf2:        buffer for second list
    chars:      number characters written into result
    chars1:     number characters read into bfl
    chars2:     number characters read into bf2
    fil:        file descriptor of resultant list
    r:          pointer to result
    result:     buffer for resultant list
    s:          pointer to bfl
    t:          pointer to bf2
*/

int except(first,second) int first,second;
/* Declaration of variables */
{
    int bfl[max_laws], bf2[max_laws], chars, chars1, chars2, fil,
        result[2 * max_laws];
    register int *r,*s,*t;

/* Fill bfl and bf2 with lawsets */
    chars1 = read(first,bfl,lawset_size);
    chars2 = read(second,bf2,lawset_size);

/* Guarantee that there is a 19999 at end (paranoia) */
    bfl[chars1 / 2] = 19999;
    bf2[chars2 / 2] = 19999;

/* Set up pointers */
    r = bfl;
    s = bf2;
    t = &result[0];

/* Since t gets pre-incremented the first one will be skipped */
    *t = 0;

```

```

while (*r < 19999)
{
    if (*r < *s) *++t = *r++;
    else
    {
        if (*r == *s) r++;
        s++;
    }
}
*++t = 19999;

/* Calculate number of characters in result */
chars = (t - &result[0]) << 1;

/* Create file to write result into */
fil = creat(temp_file_name,0666);
if (fil < 0) {perror("bool(c)"); return(-1);}

/* Write result beginning with result[1] since [0] is skipped */
write(fil,&result[1],chars);

/* Position fil at beginning of file (by closing and opening) */
close(fil);
fil = open(temp_file_name,0);

/* Increment temp_fil_name[10] since we just made a file */
temp_file_name[10] += 1;

/* Close first and second files */
close(first);
close(second);

return(fil);
}

```

```

#
#include "search.i"
/*
                                E X P R E S S I O N

        Expression is the routine for evaluating "ors".
        It is evaluated last, after "ands" and "excepts" are done.

Returns          fil          a file descriptor

Expression variables:
        fil:          file descriptor
        second:       file descriptor
*/

int expression()
{
/* Declaration of variables */
    int fil, second;

/* Call term to check for ?? */
    fil = term();
    if (fil < 0) return(fil);

    while (compar(word,"or") == 0 || compar(word,"union") == 0 ||
           compar(word,"|") == 0)
    {
        get_next_word("Or what?:");
        second = term();
        if (second < 0) return(second);
        fil = or(fil,second);
    }
    return(fil);
}

```

```

#
#include "search.i"
/*
                                F A C T O R

        Factor is the routine used for doing "ands". It is the
        highest order in the hierarchy, i.e. it gets done before "ors" or
        "excepts" regardless of which is on the line first.

Returns          fil          file descriptor to an open file
                                containing the result

Factor variables:
        fil:          file descriptor
        second:       file descriptor
*/

int factor()
{
/* Declaration of variables */
    int fil, second;

/* Call primary to get lawset */
    fil = primary();
    if (fil < 0) return(fil);

while (compar(word,"and") == 0 || compar(word,"intersect") == 0 ||
        compar(word,"&") == 0)
    {
        get_next_word("And what?:");
        second = primary();
        if (second < 0) return(second);
        fil = and(fil,second);
    }
return(fil);
}

```

```

#
#include "search.i"
/*
                                G E T   S E T

        Get_set takes a fieldname and value for that field, and looks to
        see if it is a legal value. Get_set returns -1 for illegal value,
        or a file descriptor to an open file if the value is included in
        that field.

        The "list" file is positioned at the start of the lawlist
        for that field value.

Arguments      fieldname      three character name of a field
               fvalue         a value to be looked up in that
                               field

Returns        -1             for illegal values
               fid            to an open "list" file

Get_set variables:
    bf:         buffer used for reading
    filename:   name of file to be opened
    hash_num:   hash number for fvalue
    list:       file identifier for list file
    p:          pointer used to build filenames
    r:          pointer to record read from hash table
    seek_location: location to seek to
    v:          pointer to val_record structure
    val:        file descriptor for "val" file for fieldname
    wrong:      flag = 1 if value is illegal (i.e., wrong)
*/

int get_set(field_name, fvalue) char *field_name, *fvalue;
{
    /* Declaration of variables */
    char filename[50], *p;
    int bf[100], hash_num, list, *r, seek_location, val, wrong;
    struct val_record v;

    /* Open the ".val" file for this field */
    p = concat("/cerl/celds/", field_name, filename);
    concat(".val", "", p);
    val = open(filename, 0);
    if (val < 0) {perror("getst"); return(val);}

    /* Calculate hash number for this value */
    hash_num = hash(fvalue) % max_hash_num;

    if (debugging) printf("hashes to %d\n", hash_num);
    /* Seek to proper slot in hash table -- each hash_num takes 2 bytes */
    seek_location = hash_num * 2;
    seek(val, seek_location, 0);

```



```
#
#include "search.i"
/*                                G E T _ S E T

    Get_set takes a fieldname and value for that field, and looks to
    see if it is a legal value. Get_set returns -1 for illegal value,
    or a file descriptor to an open file if the value is included in
    that field.

    The "list" file is positioned at the start of the lawlist
    for that field value.
```

Arguments	fieldname	three character name of a field
	fvalue	a value to be looked up in that field

Returns	-1	for illegal values
	fid	to an open "list" file

Get_set variables:

bf:	buffer used for reading
filename:	name of file to be opened
hash_num:	hash number for fvalue
list:	file identifier for list file
p:	pointer used to build filenames
r:	pointer to record read from hash table
seek_location:	location to seek to
v:	pointer to val_record structure
val:	file descriptor for "val" file for fieldname
wrong:	flag = 1 if value is illegal (i.e., wrong)

*/

```
int get_set(field_name, fvalue) char *field_name, *fvalue;
{
/* Declaration of variables */
    char filename[50], *p;
    int bf[100], hash_num, list, *r, seek_location, val, wrong;
    struct val_record v;

/* Open the ".val" file for this field */
    p = concat("/cerl/celds/",field_name,filename);
    concat(".val","",p);
    val = open(filename,0);
    if (val < 0) {perror("getst"); return(val);}

/* Calculate hash number for this value */
    hash_num = hash(fvalue) % max_hash_num;

if (debugging) printf("hashes to %d\n",hash_num);
/* Seek to proper slot in hash table -- each hash_num takes 2 bytes */
    seek_location = hash_num * 2;
    seek(val,seek_location,0);
```

```

/* Read from val file */
    read(val,bf,200);

/* Point r to beginning of record read */
    r = &bf[0];

/* Initialize wrong to be true */
    wrong = 1;

while (*r)
/* If *r is zero, then this hash number contains a zero in the table */
    {
        /* Locate record with this hash number */
        seek_location = ((*r * 64) + val_table_start);
        seek(val,seek_location,0);
        read(val,&v,64);
        if (debugging) printf("try %d:%s\n",*r,v.value);

        /* Compare fvalue with the value in the val file */
        wrong = compar(v.value,fvalue);

        /* If wrong is zero, we found fvalue */
        if (wrong == 0) break;

        r++;
    }

/* Close the "val" file */
    close(val);

if (wrong != 0)
/* Then fvalue was not found in the val file */
    {
        printf("searching field %s for %s\n",field_name,fvalue);
        reject_this_word("not a legal value");
        return(-1);
    }

/* Calculate location in "list" file */
    seek_location = v.first_law * 2;

/* Open ".list" file for this field */
    concat(".list","",p);
    list = open(filename,0);
    if (list < 0) {perror("getst(oli)"); return(list);}

/* Position list file at start of laws list */
    seek(list,seek_location,0);
    if (debugging) printf("get_set opens %d\n",list);

return(list);
}

```

```

#
#include "search.i"
/*
                                GET _ NEXT _ WORD

        Get_next_word is the scanner for CELDS. The argument is
        a prompt to give in case the client has not yet supplied this word.
        A special case is the nul prompt, which means the caller only wants
        the next word if it has already been supplied.
        Global variables that may be changed are:
                word:          array containing current word (nul terminated)
                request:       array containing current line of user input
                req_location:   a pointer into current position in request
                old_req_location: previous req_location

        Get_next_word recognizes punctuation and returns punctuation
        marks as a word.

        Arguments          prompt          pointer to a prompt

        Returns:           0
                           1
                           2              if word is a punctuation mark

        Get_next_word variables:
                eof:         flag indicating if user has typed an end-of-file
                w:           pointer to the "word" array
*/

int get_next_word(prompt) char *prompt;
{
/* Declaration of variables */
    char *w;
    int eof;

/* Skip over any leading blanks */
    while (*req_location++ == ' ');
    --req_location;

    if (*req_location == 0)
/* Then this is the end of the line */
    {
        do
        {
            if (*prompt == 0)
/* This is the special case of a nul prompt */
            {
                word[0] = 0;
                return(0);
            }

/* Write out prompt */
            printf("%s",prompt);

```

```
#
#include "search.i"
/*
```

L I S T E R

Lister is the procedure called by the list and print verbs to list the contents of selected laws; it only lists chosen fields.

Arguments: output_file file descriptor of output file
 out_name name of output_file
 if name is non-zero, listing will
 be sent to the line printer

Returns: -1 for error conditions
 +1 otherwise

Lister variables:

bf: buffer for reading and writing
 chars: number of characters read
 chosen: array indicating which fields to list
 flag: used to indicate when all of a field has
 been listed. This is set to the last digit
 of the field being listed.
 iobuf: buffer used only by the ggets routines
 field: field number
 laws: file descriptor of the laws file from which
 reading is taking place
 list: file descriptor of current_laws file,
 the file containing the list of law numbers
 margin: pointer to "marg_string"
 marg_string: string which is printed as the margin for
 each output line. Blanks for most lines but
 field names for first lines of fields.
 oldlawsfile: contains numerical portion of previous "laws."
 file name (which is still open). This is
 -1 if there is no open "laws." file.
 p: pointer to bf
 sel: pointer to selected
 selected: array containing law numbers of laws to be printed
 toc: file descriptor of "laws.toc" file
 type_start: array containing the "laws.toc" record for the
 current law number

```
*/
```

```
int lister(output_file,out_name) int output_file; char *out_name;
{
int *aborter();
/* Declaration of variables */
int chars, chosen[14], field, laws, list, oldlawsfile, *sel,
selected[max_laws], toc, type_start[13];
char bf[122], flag, iobuf[550], *margin, marg_string[10], *p;
```

```

    /* Read response from the terminal */
    eof = gets(request);

    if (eof == 0)
    /* Then user typed a cntrl-d */
    {
        word[0] = 0;
        reading_commands = 0;
        return(0);
    }

    /* Reset req_location */
    req_location = &request[0];

    /* Get rid of leading blanks */
    while (*req_location++ == ' ');
    --req_location;

    if (*req_location == '!')
    /* Then this is a UNIX command */
    {
        req_location++;
        execute(req_location);
        /* Set to zero so we loop again */
        *req_location = 0;
    }
    } while (*req_location == 0);
}

/* Reset old_req_location */
old_req_location = req_location;

/* Check for punctuation in word */
w = &word[0];
if (punctuation[*req_location & 0177])
/* Then there is recognized punctuation */
{
    *w++ = *req_location++;
    *w = 0;
    return(2);
}

/* Word must be alphabetic, so copy to next blank or punctuation */
while (punctuation[*req_location & 0177] == 0)
    *w++ = *req_location++;

/* Set word[0] to zero */
*w = 0;

return(1);
}

```

```

/* Initialization */
    oldlawsfile = -1;
    for (field = 1; field < 14; field++) chosen[field] = 0;

/* Open current laws file */
    list = open("current_laws",0);
    if (list < 0) {reject_this_word("no laws selected"); return(-1);}

/* Fill in selected array */
    chars = read(list,selected,lawset_size);
    close(list);
    selected[chars / 2] = 19999;
    if (selected[0] >= 19999)
    {
        reject_this_word("no laws selected");
        return(-1);
    }

/* Call get_next_word to find out which fields to list */
    get_next_word("What field(s)?");

while (word[0] != 0)
/* We still have more fields requested on this line of input */
    {
        if (compar(word,"all") == 0)
        {
            for (field = 2; field < 11; field++) chosen[field] = 1;
            get_next_word("");
            break;
        }
        if (compar(word,"abort") == 0)
        {
            reject_this_word("listing aborted");
            return(-1);
        }
        field = xlate_field(word);
        if (field < 0) {reject_this_word("not a field name"); return(-1);}
        /* Fill in the chosen array for requested fields, except
           if keywords are requested, show topics */
        if (field != 13) chosen[field] = 1;
        else chosen[12] = 1;

        get_next_word("");
        /* Check for optional syntax */
        if (compar(word,",") == 0) get_next_word("Next field?");
        if (compar(word,"and") == 0) get_next_word("Next field?");
    }

/* If this is a print batch it off */
if (*out_name)
/* Out_name will have a value of zero for the terminal */

```

```

    {
        if (fork() != 0) return(1);
    }

else
/* Set a trap for interrupt */
signal(2,aborter);

/* Open table of contents (toc) file */
toc = open("/cerl/celds/laws.toc",0);
if (toc < 0)
    {
        perror("lister(otoc)");
        if (*out_name) exit();
        else return(-1);
    }

sel = &selected[0];
while (*sel < 19999)
/* While there are still law numbers left */
    {
        /* Put two newlines at beginning of bf */
        p = &bf[0];
        *p++ = 012;
        *p++ = 012;

        /* Put "law" number, and another newline into bf */
        p = concat("law ",locv(0,*sel),p);
        *p++ = 012;

        write(output_file,bf,(p - &bf[0]));

        /* Locate appropriate record in toc file */
        lseek(toc,(*sel * 26.0));
        chars = read(toc,type_start,26);
        if (chars < 0)
            {
                perror("lister(rlw)");
                close(toc);
                if (oldlawsfile != -1) close(laws);
                break;
            }
        if (type_start[0] != oldlawsfile)
        /* Then we need to open a new "laws." file */
            {
                if (oldlawsfile != -1) close(laws);
                concat("/cerl/celds/laws.",locv(0,type_start[0]),bf);
                laws = gopen(bf,&iobuf);
                if (laws < 0) {perror("lister"); close(toc); break;}
                oldlawsfile = type_start[0];
            }
    }

```

```

for (field = 1; field < 13; field++)
{
    if (chosen[field])
    /* Then field has been requested */
    {
        /* Construct margin */
        margin = &marg_string[0];
        concat(fld_name[field],":",margin);

        gseek(&iobuf,type_start[field]);
        chars = ggets(bf,&iobuf);
        flag = bf[7];
        while (flag == bf[7] && chars > 0)
        {
            p = concat(margin,&bf[10],bf);
            margin = " ";
            *p++ = 012;
            write(output_file,bf,(p - &bf[0]));
            if (reading_commands != 1)
            /* Then aborter was called */
            {
                close(toc);
                close(laws);
                printf("\n\n\nlisting aborted\n");
                return(-1);
            }
            chars = ggets(bf,&iobuf);
        }
    }
    sel++;
}

/* Write four newlines between laws */
p = &bf[0];
*p++ = 012;
*p++ = 012;
*p++ = 012;
*p++ = 012;
write(output_file,bf,4);

/* Close files */
close(toc);
close(laws);

/* This code is only for the child process (print). Give the
listing to the lineprinter (lpr) and exit. */
if (*out_name)
/* Then this is the child */
{
    close(output_file);
}

```

```
concat("lpr -r ",out_name,bf);  
execute(bf);  
exit();  
}
```

```
return(1);
```

```
}
```

```
#
#include "search.1"
/*
```

O R

Or takes two lists of law numbers and logically "ors" the two into a third list, i.e. it make a composite list of laws occurring in either list.

Arguments	first	file descriptor of file containing first list of law numbers
	second	file descriptor of file containing second list of law numbers

Returns	fil	file descriptor of file containing the "ored" list
---------	-----	--

Or variables:

bfl:	buffer for first list
bf2:	buffer for second list
chars:	number characters written into result
chars1:	number characters read into bfl
chars2:	number characters read into bf2
fil:	file descriptor of resultant list
r:	pointer to result
result:	buffer for resultant list
s:	pointer to bfl
t:	pointer to bf2

*/

```
int or(first,second) int first,second;
```

```
/* Declaration of variables */
```

```
{
    int bfl[max_laws], bf2[max_laws], chars, chars1, chars2, fil,
        result[2 * max_laws];
    register int *r,*s,*t;
```

```
/* Fill bfl and bf2 with lawsets */
```

```
chars1 = read(first,bfl,lawset_size);
chars2 = read(second,bf2,lawset_size);
```

```
/* Guarantee that there is a 19999 at end (paranoia) */
```

```
bfl[chars1 / 2] = 19999;
bf2[chars2 / 2] = 19999;
```

```
/* Set up pointers */
```

```
r = bfl;
s = bf2;
t = &result[0];
```

```
/* Since t gets pre-incremented the first one will be skipped */
```

```
*t = 0;
```

```

do
{
    if (*r < *s) *++t = *r++;
    else
    {
        if (*r == *s) r++;
        *++t = *s++;
    }
} while (*t < 19999);

/* Calculate the number of characters in result */
chars = (t - &result[0]) << 1;

if (chars > lawset_size)
/* Then our lawset is too big for the buffers in CELDS */
{
    printf("lawset exceeds %d laws\n", (max_laws - 1));
    printf("only the first %d laws will be used\n", (max_laws - 1));
    t = &result[max_laws];
    *t = 19999;
    chars = lawset_size;
}

/* Create fil to write result into */
fil = creat(temp_file_name, 0666);
if (debugging) printf("or creating %s\n", temp_file_name);
if (fil < 0) {perror("bool(c)"); return(-1);}

/* Write result beginning with result[1] since [0] is skipped */
write(fil, &result[1], chars);

/* Position fil at beginning of file (by closing and opening) */
close(fil);
fil = open(temp_file_name, 0);
if (debugging) printf("or %d with %d giving %d (%d laws)\n",
    first, second, fil, (t - &result[1]));

/* Increment temp_file_name since we just made a file */
temp_file_name[10] += 1;

/* Close first and second files */
close(first);
close(second);

return(fil);
}

```

```

#
#include "search.i"
/*
PRIMARY

Primary finds the lowest order terms in expressions for
commands. This is the "value" of the field requested. Primary
returns -1 on various error conditions. Otherwise it creates a file
containing the set of laws with the specified value.

Returns      fil      file descriptor of an open file
                  containing the set of laws with the
                  current value
              -1      on error conditions

Primary variables:
  content:      array used to put an accession number in to
                  write to a file
  delim:       delimiter for values, either a nul or a
                  double quote mark depending on whether a
                  quote has been encountered
  fil:         fil descriptor of current lawset
  fld:         flag indicating whether a field is being searched
  fld_num:     field number
  p:           pointer
  q:           pointer
  st:          indicates if a set name is being searched
  value:       array containing the current value
*/

int primary()
{
/*Declaration of variables */
  int content[2], fil, fld, fld_num, st;
  char delim, *p, *q, value[62];

/* Check for ordering by parenthesis */
  if (compar(word,"(") == 0)
  {
    get_next_word("Search criterion:");
    fil = expression();
    if (compar(word,")") == 0)
    {
      get_next_word("");
      return(fil);
    }
    reject_this_word("right paren expected");
    return(-1);
  }

  fld = 0;
/* Remove optional text in command structure */

```

```

if (compar(word,"all") == 0)
{
    get_next_word("Field name?:");
    fld = 1;
}.....
if (compar(word,"laws") == 0)
{
    get_next_word("Field name?:");
    fld = 1;
}
if (compar(word,"with") == 0)
{
    get_next_word("Field name?:");
    fld = 1;
}
if (compar(word,"where") ==0)
{
    get_next_word("Field name?:");
    fld = 1;
}

/* Translate field name to a number */
fld_num = xlate_field(word);

if (field_is_searchable[fld_num])
/* Then this is a searchable field */
{
    get_next_word("What value?:");
    if (compar(word,"is") == 0) get_next_word("What value?:");
    if (*word == '"')
    /* Then there is a term in quotes */
    {
        /* Skip past quote mark */
        old_req_location++;

        /* Set delimiter to look for other quote */
        delim = '"';

        request[121] = '"';
    }
    else delim = 0;

    /* Copy word into value until delim */
    p = &value[0];
    q = old_req_location;
    while (*q != delim) *p++ = *q++;

    if (q > &request[120])
    /* Then we never found the delimiter */
    {
        reject_this_word("closing quote mark missing");
    }
}

```

```

        return(-1);
    }

    *p = 0;
    if (*q != 0) q++;
    req_location = q;
    old_req_location = q;
    get_next_word("");
    if (fld_num == 1)
    /* Then search was for accession number */
    {
        /* Convert accession number to integer */
        content[0] = atoi(value);

        /* put in a test here to see if acc num is legal */
        /* Create temp_file */
        fil = creat(temp_file_name,0666);
        if (fil < 0) {perror("getacc"); return(fil);}

        content[1] = 19999;
        write(fil,content,4);
        /* Position fil at beginning of file */
        close(fil);
        fil = open(temp_file_name,0);

        /* Increment temp_file_name */
        temp_file_name[10]++;

        return(fil);
    }

    if (compar(value,"abort") == 0)
    {
        reject_this_word("search aborted");
        return(-1);
    }

    /* Get the set of laws for this field and value */
    fil = get_set(fld_name[fld_num],value);

    return(fil);
}

if (fld_num > 0)
{
    reject_this_word("field is not a searchable field");
    return(-1);
}

if (compar(word,"abort") == 0)
{

```

```

        reject_this_word("search aborted");
        return(-1);
    }
    if (fld)
    {
        reject_this_word("field name expected");
        return(-1);
    }

    /* If we have reached this point, then it must be a setname */

    st = 0;
    /* Remove optional text */
        if (compar(word,"my" == 0))
        {
            get_next_word("Lawset name?:");
            st = 1;
        }
        if (compar(word,"set" == 0))
        {
            get_next_word("Lawset name:");
            st = 1;
        }
    if (compar(word,"set") == 0) {get_next_word("Lawset name?:"); st = 1;}
    if (compar(word,"abort") == 0)
    {
        reject_this_word("search aborted");
        return(-1);
    }

    /* File will have an arbitray "_" preceding the name */

    concat("_",word,value);
    fil = open(value,0);

    if (fil < 0)
    /* Then this file does not exist */
        if (st == 0)
            reject_this_word("Neither a lawset name nor a field name");
        else reject_this_word("no set by that name");

    get_next_word("");

    return(fil);
}

```

```

#
#include "search.1"
/*
                                R E J E C T _ T H I S _ W O R D

    Reject_this_word is the CELDS error processing routine.

Arguments      error_message  pointer to message to be printed

Globals:
    req_location:  pointer to current position in request
    word:         array containing the current word

Reject_this_word variables:
    error_message: pointer to message to be printed
*/

int reject_this_word(error_message) char *error_message;
{
    printf("ERROR: %s - %s\n",word,error_message);
    /* Reset global indicators */
        *req_location = 0;
        word[0] = 0;
}

```

```

#
#include "search.i"
/*                               S H O W

        Show writes the accession numbers of "current_laws" on
        the terminal, ten to a line.

Returns:      -1      for error conditions (no laws selected).
              0      otherwise

Show variables:
      bf:      buffer used for reading
      chars:   number of characters read
      fil:     file descriptor of current_laws
      i:       counter
      *r:      pointer to bf
*/

int show()
{
int *aborter ();
/* Declaration of variables */
    int bf[max_laws], chars, fil, i, *r;

/* Open current_laws file */
    fil = open("current_laws",0);
    if (fil < 0)
    {
        reject_this_word("no laws selected");
        return(-1);
    }

/* Read file into bf */
    chars = read(fil,bf,lawset_size);

close(fil);
/* Every two bytes is an integer number. Put a 19999 at end. */
    bf[chars / 2] = 19999;
    if (bf[0] >= 19999)
    {
        reject_this_word("no laws selected");
        return(-1);
    }

    r = &bf[0];
/* Set trap for interrupt signal */
    signal(2,aborter);

while (*r < 19999)
/* Write out list, ten to a line */
    {

```

```

    for (i = 0; i < 9; i++)
    {
        if (*r >= 19999) break;
        printf("%5d ",*r++);
    }
    printf("\n");
    /* Check to see if aborter was called (reading_commands = 2) */
    if (reading_commands != 1) return(-1);
}
/* Prepare next "word" for celds */
get_next_word("");
/* Throw away optional words on "show" command */
if (compar(word,"the") == 0) get_next_word("");
if (compar(word,"laws") == 0) get_next_word("");
return(0);
}

```

```

#
#include "search.i"
/*
                                T E R M

        Term is the routine for evaluating "excepts". It comes
second in the hierarchy. "Ands" are evaluated first, "excepts"
second, and "ors" last.

Returns          fil          file descriptor of result file

Term variables:
        fil:          file descriptor
        second:       file descriptor
*/

int term()
{
/* Declaration of variables */
    int fil, second;

    fil = factor();
    if (fil < 0) return(fil);

    while (compar(word,"except") == 0)
    {
        get_next_word("Except what?:");
        second = factor();
        if (second < 0) return(second);
        fil = except(fil,second);
    }
    return(fil);
}

```

```

#
#include "search.i"
/*                                X L A T E _ F I E L D

        Xlate_field translates a field_name to a fld_num, and returns
        the fld_num. If the field_name is not the name of a legal field,
        then -1 is returned.

        For matching a field to a number, only the first three
        characters of field_name are used.

Arguments:      field_name      name of field

Returns:        -1              if not a legal field
                fieldnum        for recognized fields

Globals:        (see globals for "expr.c")

Xlate_field variables:
                fld_num:        field number
                p:
                short_fname[4]:  first 3 characters of field_name
*/

int xlate_field(field_name) char *field_name;
{
/* Declaration of variables */
    char *p, short_fname[4];
    int fld_num;

/* Put first 3 characters of field_name into short_fname */
    p = &short_fname[0];
    *p++ = *field_name++;
    *p++ = *field_name++;
    *p++ = *field_name;
    *p = 0;

    for (fld_num = 1; fld_num < 14; fld_num++)
        if (compar(fld_name[fld_num],short_fname) == 0) return(fld_num);

    return(-1);
}

```

Appendix H - General Utility Subroutines

/*

COMPAR

Compar compares two strings and determines if they are the same.

Arguments: s1 pointer to string one
 s2 pointer to string two

Returns: 0 if strings are the same
 - if string one is "bigger"
 + if string two is "bigger"

Compar variables:

 greater: algebraic difference of chars in s1 - s2
 p: pointer to s1
 q: pointer to s2

*/

```
int compar(s1,s2) char *s1,*s2;
{
/* Declaration of variables */
    register char *p,*q;
    register int greater;

    p = s1;
    q = s2;
    while ((greater = *p - *q++) == 0 && *p++ != 0);
    return (greater);
}
```

/*

C O N C A T

Concat concatenates two strings and returns the composite string.

Arguments:	first	pointer to first string
	second	pointer to second string
	result	pointer to end of resulting string

Returns: pointer to end of result string

*/

```
char *concat(first, second, result) char *first, *second, *result;
```

```
{
```

```
/* Put first string into result string */  
while (*result++ = *first++);
```

```
/* Back up over nul */  
--result;
```

```
/* Put second string into result */  
while (*result++ = *second++);
```

```
/* Back up over nul */  
--result;
```

```
return(result);
```

```
}
```

```

    char *gnextp;
    char *gstop;
    int geof;
    char gdbuf[513];
};

int ggets(bf, stru) char *bf; struct gstru *stru;
{
    /* Declaration of variables */
    int ch, chars, more_to_read;
    register char *p, *q;

    /* Point q to the start of bf */
    q = bf;

    /* Continue this as long as no new-line has been found, other
       than the artificial one at gdbuf[512] */
do    {
        /* Reset new-line flag */
        more_to_read = 0;

        /* point p to next character to be read */
        p = stru->gnextp;

        /* copy next line into the callers buffer, bf */
        while (*p != 012) *q++ = *p++;

        /* Now, adjust gnextp to point at next character to be read */
        stru->gnextp = p + 1;

        if (p >= stru->gstop)
        /* Then we need to read into the next block */
        {
            /* Turn on no new-line indicator */
            more_to_read = 1;

            /* Check for end-of-file */
            if (stru->geof) {bf[0] = 0; return(0);}

            /* Point gnextp to beginning of gdbuf */
            stru->gnextp = &stru->gdbuf[0];

            /* Read next block from file to gdbuf */
            ch = read(stru->gfildes, stru->gdbuf, 512);

            if (ch < 0)
            /* Then we can't read a block and haven't yet
               encountered a new-line */
            {
                perror("ggets(read)");
            }
        }
    }
}

```

/*

G G E T S

The gopen, ggets, and gseek routines are an alternative way to read ASCII files. They are designed primarily for sequential line-oriented reading, with some seeks. The only seeks allowed are absolute character number (ptrname = 0).

A core buffer (gdbuf) is filled by the gopen routine with one block of data from a file. When ggets is called, data is transferred from gdbuf to the caller's buffer (bf) until a new-line is encountered. If no new-line is encountered by the end of the block, the next block of data is read from the file into gdbuf and the transfer continues.

To use ggets with a file, the file must be opened with gopen and seeks should be performed using gseek.

Globals: (for ggets, gopen, and gseek)

gstru: This is a structure which is used for reading lines from a file. Its components are:

gdbuf	--	a buffer holding one block of data.
geof	--	0 until last block is read, 1 when gdbuf contains last block
gfildes	--	file descriptor
gnextp	--	a pointer to the next character to be read from gdbuf
gstop	--	a pointer to the next critical character, signaling when to read a new block. After eof, this points to the last character of data.

Arguments:

bf:	a pointer to the caller's buffer into which the next line of data should be read
stru:	a pointer to the structure (array) of shape gstru which contains file descriptor, etc.

Returns: the number of characters read into bf

Ggets variables:

ch:	number of characters read from file to gdbuf
chars:	number of characters read into bf
more_to_read:	flag indicating if read continues into the next block
p:	a pointer into the core buffer, gdbuf
q:	a pointer into the caller's buffer, bf

Returns:

*/

```
struct gstru
{
    int gfildes;
```

/*

G O P E N

Gopen opens a file and reads the first block of data into a core buffer, and sets up geof, gnextp, and gstop. The file is opened for reading only.

Globals: see list for ggets
 (gopen should only be used when using ggets.)

Arguments: fname name of the file to be opened
 stru pointer to a structure of shape gstru
 which will contain the core buffer and
 parameters for this file

Returns: the file descriptor of the file opened

Gopen variables:

 ch: number of characters read from file to gdbuf

*/

```
int gopen(fname, stru) char *fname; struct gstru *stru;
{
/* Declaration of variables */
    int ch;

/* Open file, mode 0 */
    stru->gfildes = open(fname, 0);
    if (stru->gfildes < 0)
    {
        perror("gopen");.....
        return(stru->gfildes);..
    }

/* Read first block from file into gdbuf */
    ch = read(stru->gfildes, &stru->gdbuf[0], 512);
    if (ch < 0) {perror("gopen(read)"); return(ch);}

/* Set up stop, nextp, and eof */
    stru->gstop = &stru->gdbuf[512];
    stru->gnextp = &stru->gdbuf[0];
    stru->geof = 0;

    if (ch < 512)
/* Then this is the last (and only) block */
    {
        stru->geof = 1;
        stru->gstop = &stru->gdbuf[ch];
        *stru->gstop = 012;
    }

/* Put in the terminating new-line */
```

```

        stru->geof = 1;
        *stru->gnextp = 012;
        stru->gstop = stru->gnextp;
        bf[0] = 0;
        return(0);
    }

    if (ch < 512)
    /* Then this must be the last block */
    {
        stru->geof = 1;
        stru->gstop = &stru->gdbuf[ch];
        *stru->gstop = 012;
    }

    /* make absolutely sure there is a newline to find */
    stru->gdbuf[512] = 012;

    }
    } while(more_to_read);

    /* Put nul at end of bf */
    *q++ = 0;

    /* Calculate characters transferred to bf */
    chars = q - bf;

    return(chars);
}

```

```
    stru->gdbuf[512] = 012;  
    return(stru->gfildes);  
}
```

/*

G S E E K

Gseek seeks an absolute character number in a file opened by gopen, fills gdbuf, and adjusts nextp, stop, and eof accordingly.

Globals: see list for ggets
 (gseek should only be used when using ggets.)

Arguments: offset character number desired
 stru pointer to a structure of shape gstru
 which will contain the core buffer and
 parameters for this file

Returns: 1 if seek was successful
 error code, otherwise

Gopen variables:

 ch: number of characters read from file to gdbuf

*/

a

```
int gseek(stru,offset) struct gstru *stru; int offset;
```

```
{
```

```
/* Declaration of variables */
```

```
    int ch;
```

```
/* Seek to offset */
```

```
    ch = seek(stru->gfildes,offset,0);
```

```
    if (ch < 0) {perror("gseek(seek)"); return(ch);}
```

```
/* Read block of data into gdbuf */
```

```
    ch = read(stru->gfildes,&stru->gdbuf[0],512);
```

```
    if (ch < 0) {perror("gseek(read)"); return(ch);}
```

```
/* Set up stop, nextp, and eof */
```

```
    stru->gstop = &stru->gdbuf[512];
```

```
    stru->gnextp = &stru->gdbuf[0];
```

```
    stru->geof = 0;
```

```
if (ch < 512)
```

```
/* Then this is the last block */
```

```
{
```

```
    stru->geof = 1;
```

```
    stru->gstop = &stru->gdbuf[ch];
```

```
    *stru->gstop = 012;
```

```
}
```

```
/* Put in the terminating new-line */
```

```
    stru->gdbuf[512] = 012;
```

```
return(1);
```

```
}
```

Hash encrypts a string into a single integer number.

Arguments: str pointer to string to be encrypted

Returns: encrypted integer number

Hash variables:

total: running total for additions in encryption

*/

```
int hash(str) char *str;
```

```
{
```

```
/* Declaration of variables */
```

```
register int total;
```

```
/* Initialize total */
```

```
total = 0;
```

```
/* Add up words as integer numbers */
```

```
while (*str) total += *str++;
```

```
return(total);
```

```
}
```

/*

L S E E K

Lseek seeks an absolute character number in a file. The number may be greater than 65535.

Arguments:	fil	--	integer file descriptor
	dcharnum	--	double, character number sought
Returns:	1		if no block seek performed
	2		if block seek performed

Lseek variables:

block: block number of desired character number
dcharnum: desired character number (double)
fil: file descriptor
plus: position of desired character number in its block
seeker: desired character number (long)

*/

```
int lseek(fil,dcharnum) int fil; double dcharnum; {  
/* Declaration of variables */
```

```
    int block, plus;  
    long seeker;
```

```
/* convert dcharnum to long */  
    seeker = dcharnum;
```

```
    if (dcharnum < 65535.0)  
/* Then we don't need a block seek */  
    {  
        plus = seeker;  
        seek(fil,plus,0);  
        return(1);  
    }
```

```
    else  
/* A block is sought first */  
    {  
        block = seeker / 512;  
        plus = seeker % 512;  
        seek(fil,block,3);  
        seek(fil,plus,1);  
        return(2);  
    }  
}
```

```
/*
```

MOVE

Move moves a specified number of characters from one location to another.

Arguments: count number of characters to be moved
 from pointer to string to be moved
 to pointer to receiving location

Variables:

 i: counter

```
*/
```

```
int move(from,to,count) char *from, *to; int count;
```

```
{
```

```
/* Declaration of variables */
```

```
    int i;
```

```
for (i = 0; i < count; i++) *to++ = *from++;
```

```
return;
```

```
}
```

/*

SEARCH . I

Search.i is the include file of structures and defines used to access the search file x.val.

The value file contains the names of all of the values for a given field and a hash table for quick access to those names. The numbers of the laws that belong in a given set are found in a separate file named x.list (where x is the field name).

The value file starts with the hash table for max_hash_num + 10 words. The hash table contains 1 word per hash number, either 0 or the value_number of a value that hashes here. If a slot is full, and the value is different from the value to be inserted, the hash_number is simply incremented by one. This kind of hash table, while exceedingly simple to build, tends to get cluttered when it is over 60% full. For this reason, max_hash_num is chosen to be a large number.

Each set has a value number greater than 0. The next N records (one per set) are 64 characters long. Each record contains 62 characters for the set name, and an integer telling where in the list file the laws list for that set starts.

The list file is simply lists of law numbers terminated by 19999.

*/

```
#define hash_size    4011
#define lawset_size 3000
#define max_hash_num 4001
#define max_laws     1501
#define val_table_start 7958
```

```
struct val_record
{
    char value[62];
    int first_law;
};
```

```
/* Declaration of global variables for the retriever */
char *fld_name[14], *old_req_location, *req_location;
char field_is_searchable[14];
char punctuation[128], request[122], temp_file_name[12], word[62];
int client, debugging, reading_commands, requests;
```

CERL DISTRIBUTION

Chief of Engineers
ATTN: DAEN-MCZ-S
ATTN: DAEN-ASI-L (2)
ATTN: DAEN-RDL
ATTN: DAEN-MCE-D
Dept of the Army
WASH DC 20314

Defense Documentation Center
ATTN: DDA (12)
Cameron Station
Alexandria, VA 22314

Welsh, Rikki L

System documentation for Computer - aided Environmental Legislative Data System. - Champaign, IL : Construction Engineering Research Laboratory ; Springfield, VA : available from National Technical Information Service, 1978.

159 p. ; 27 cm. (Special report - Construction Engineering Research Laboratory ; N-31)

1. Environmental law - data processing. I. Title. II. Title : Computer - aided Environmental Legislative Data System. III. Series : U.S. Construction Engineering Research Laboratory. Special report ; N-31.